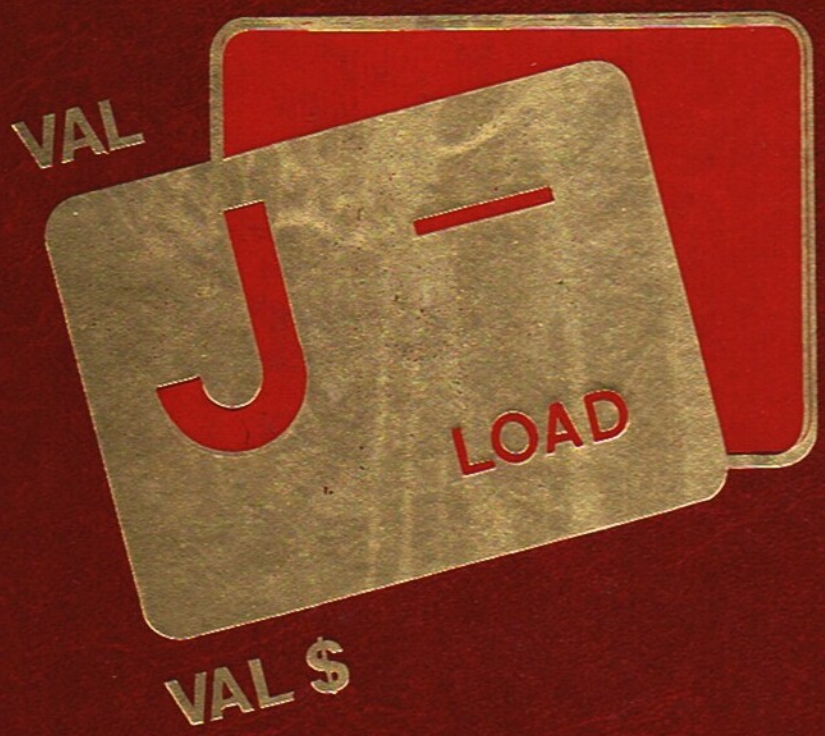
RUN

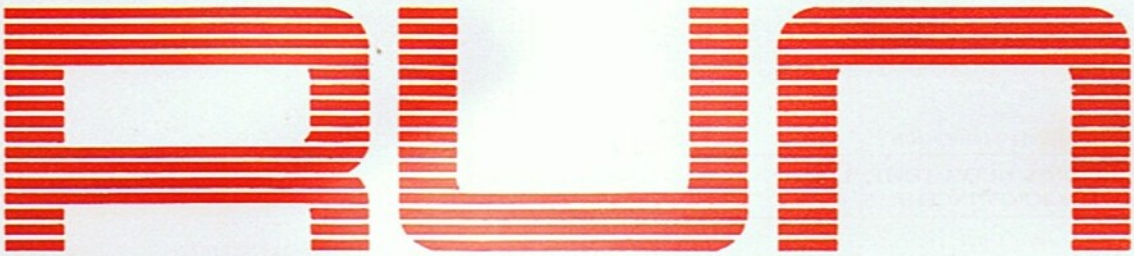
Enciclopedia Práctica del Spectrum



Nueva Lente/Ingelek

RUN

Enciclopedia Práctica del Spectrum



Enciclopedia Práctica del Spectrum

Nueva Lente/Ingelek

Una publicación

EDICIONES NUEVA LENTE, S. A.
y EDICIONES INGELEK, S. A.

Director-editor por Nueva Lente, S. A.:
MIGUEL J. GONI
Director-editor por Ingelek, S. A.:
ANTONIO M. FERRER
Director de producción:
RICARDO ESPAÑOL
Jefe de producción:
SANTOS ROBLES

Director de la obra:
FERNANDO LOPEZ MARTINEZ
Coordinación:
VICENTE ROBLES
Colaboradores:
JUAN MANUEL LOPEZ MARTINEZ
CARLOS DE LA OSSA
JUAN MANUEL MAYORAL

Diseño Gráfico:
JOSE OCHOA
Maquetación:
CARLOS GONZALEZ-AMEZUA
Fotografía:
«Equipo Gálata»
(ALBINO LOPEZ y EDUARDO AGUDELO)
Ilustraciones:
JOSE OCHOA
ALFONSO MENDEZ
ANTONIO PERERA

© Ediciones Nueva Lente, S. A.
Madrid 1985
Dirección, redacción y administración:
Benito Castro, 12. 28028-Madrid
Teléfs.: 245 45 98 y 246 73 67

© Ediciones Ingelek, S. A.
Madrid 1985
Dirección, redacción y administración:
Avda. Alfonso XIII, 141. 28016-Madrid
Teléf.: 250 58 20

ISBN tomo primero: 84-7534-120-9
ISBN tomo segundo: 84-7534-121-7
ISBN tomo tercero: 84-7534-122-5
ISBN tomo cuarto: 84-7534-123-3
ISBN de la obra: 84-7534-118-7

Fotomecánica: OCHOA, S. A.
Ricardo Ortiz, 74. 28017-Madrid

Impresión: GRAFICAS REUNIDAS, S. A.
Avda. de Aragón, 56. 28027-Madrid

Depósito legal: M-9896-1985
PRINTED IN SPAIN

Queda prohibida la reproducción total o parcial de esta obra sin permiso escrito de los editores.

EL AZAR INFORMATIZADO

ARROJAR una moneda al aire, es una práctica común para la resolución de situaciones conflictivas: quién paga la merienda, quién inicia el partido, y otras muchas de extraordinaria trascendencia. Se utiliza en definitiva el azar como factor de decisión, pues nos parece algo absolutamente involuntario. Sin embargo, una ciencia tan exacta como las matemáticas, también se ha ocupado del estudio de la «suerte», y mediante el CALCULO DE PROBABILIDADES, aunque no es capaz de asegurarnos si saldrá cara o cruz, sí puede informarnos de nuestra probabilidad de empezar el partido o pagar la merienda, aunque evidentemente, puede ser que el azar nos juegue una mala pasada.

Al tipo de sucesos que dependen de la suerte, como los que hemos visto hace un momento, se les conoce como SUCESOS ALEATORIOS, y el BASIC es el único lenguaje de programación, entre los de mayor difusión, que tiene capacidad para emularlos; en pocas palabras, el BASIC es capaz de arrojar una moneda al aire, tirar un dado, o darle la vuelta a un bombo de lotería. Más adelante comprobaremos, gracias al ordenador, cómo los sucesos aleatorios siguen determinadas tendencias, que pueden ser estudiadas matemáticamente (probabilidades); pero antes vamos a observar, la forma en que el lenguaje de programación trata los sucesos aleatorios.

Como en la mayoría de los casos, el tratamiento que el ordenador da a la «suerte», es numérico, es decir, la habilidad aleatoria del BASIC se concreta en la generación de números al azar. Esto es suficiente, y es tarea del programador interpretar estos números que la máquina proporciona. Así por ejemplo, si queremos simular la tirada de un dado, pediremos a nuestro Spectrum que encuentre un número natural (entero y sin decimales) al azar entre 1 y 6; si lo que queremos es arrojar una moneda al aire, el número podría ser 1 ó 2 (1=Cara, 2=Cruz), etc...

Aparentemente, no debería ser nada difícil para un ordenador producir un número aleatorio, dada su extraordinaria facilidad para la realización de complejos cálculos numéricos, pero si nos fijamos un poco, observaremos que los resultados de todos esos cálculos y operaciones complejas a que nos referimos, son siempre muy concretos y en absoluto aleatorios.

La técnica para obtener un número al azar, como el que produce el BASIC, es bastante sofisticada. Dos son las sentencias que controlan el proceso de generación de números aleatorios: **RANDOMIZE** (abreviado en el teclado a **RAND**) y **RND**. Antes de entrar en materia, hemos de hacer una pequeña aclaración sobre lo «trucados» que están los dados y monedas del BASIC.

PSEUDOALEATORIEDAD

Bajo el complicado nombre con que titulamos este epígrafe, se esconde una característica importante de los números que el BASIC genera al azar. Estos números, se denominan PSEUDOALEATORIOS, es decir, PROXIMOS a la aleatoriedad, porque dado su método de obtención, no dependen completamente de la suerte.

Básicamente, para la extracción de números aleatorios, el Spectrum toma un número inicial, denominado SEMILLA (en inglés *seed),* a partir del cual realiza una serie de cálculos de relativa complejidad, que desfiguran este valor inicial; sin embargo, dada la exactitud matemática, a una misma semilla siempre corresponde un mismo número pseudoaleatorio.

En cualquier caso, la definición de pseudoaleatorio es bastante estricta, puesto que, aunque a no-

Una máquina tan precisa como nuestro Spectrum, también puede simular ciertos sucesos que se producen al azar.

¡!

El algoritmo para la obtención de números naturales aleatorios entre límites, es de gran importancia: **R=A+INT (RND*B);** donde R es el número aleatorio natural, A el límite inferior y B la diferencia entre el límite superior y el inferior. Tengamos en cuenta que en esta fórmula el límite superior sí se alcanza.

*

Los números que se obtienen mediante la función **RND** no son completamente aleatorios, aunque en la práctica se pueden considerar como tales.

El azar informático no se puede considerar completamente aleatorio.

sotros nos parezca que la generación de números es realmente fruto exclusivo de la casualidad, no menos que una auténtica lotería, se trata realmente de la selección de números de una serie de 65536 semillas, bastante grande para aparentar que los números no se repiten pero, desde luego, no infinita.

Una vez conocidas las limitaciones que la informática presenta en el campo de la aleatoriedad, pasaremos a estudiar el formato en que el Spectrum nos proporciona los números al azar. Estos se encuentran siempre comprendidos entre 0 y 1, de la forma: **0 <= R < 1;** siendo **R** un número real comprendido en el intervalo, que puede llegar a adoptar el valor mínimo (0) pero no el máximo (1).

Sin embargo, en la mayoría de los casos, debemos elegir un número entre 1 y 90 (bingo), entre 1 y 6 (tirada de un dado), entre 2 y 12 (tirada de dos dados), entre 1 y 40 (extracción de una carta de la baraja española), etc... ¿Cómo solucionamos entonces el problema?

Puesto que el BASIC nos suministra un número decimal cada vez que recurrimos a la generación aleatoria, nos es necesario introducir algunos cálculos para conseguir que la secuencia de números se produzca entre determinados valores, generalmente dentro del conjunto de los naturales (positivos no decimales, incluyendo el cero). Ahora que ya disponemos de los conocimientos básicos necesarios, entraremos en el estudio de la forma en que el BASIC efectúa la extracción de números aleatorios, y más concretamente, en cómo introducir el factor suerte en nuestros propios programas.

LA FUNCION RND

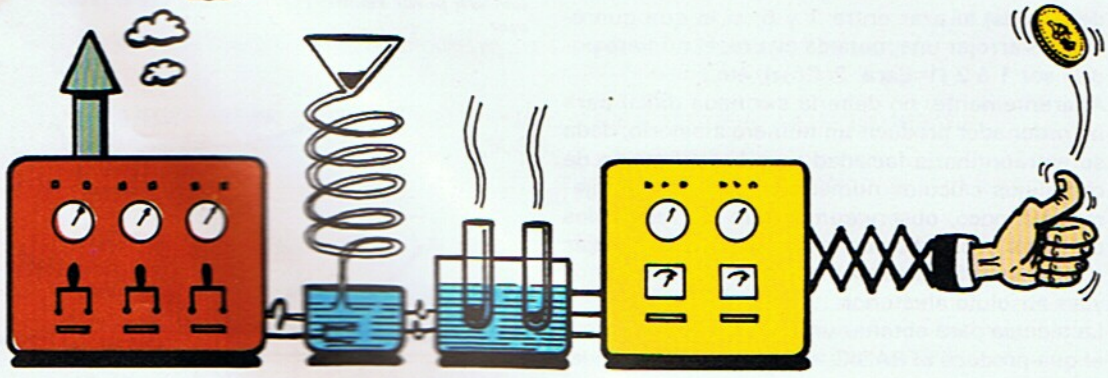
La función BASIC encargada de la generación de números aleatorios es **RND,** y carece de argumentos. Basta con recurrir a ella por medio de la sentencia **LET,** si queremos almacenar su resultado, o con **PRINT,** si lo que deseamos es imprimir directamente en la pantalla. Aunque estos dos sistemas citados son los más habituales, es obvio que puede ser utilizada como cualquier función BASIC.

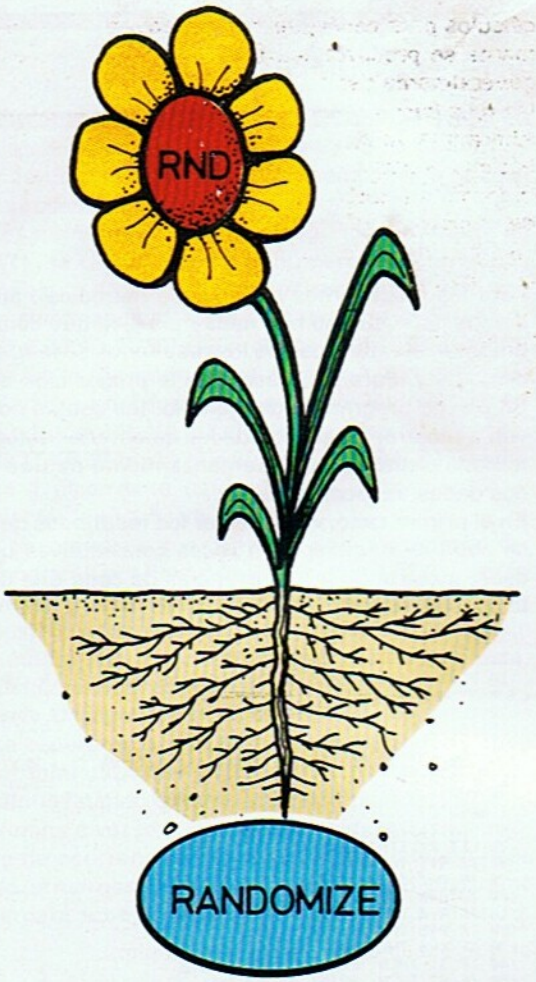
Pero, para comprender gráficamente el aspecto de los números que produce esta función, es mejor que pasemos al siguiente ejemplo:

```
10 REM - GENERACION ALEATORIA
20 FOR I=1 TO 44
30 PRINT RND,;
40 NEXT I
```

Con este programa, llenamos la pantalla de una verdadera sopa de «números», en lugar de la tradicional sopa de «letras». Como ejercicio mental, podemos plantearnos el buscar alguna relación lógica entre los resultados obtenidos, aunque nuestro esfuerzo será en vano. La serie es lo suficientemente complicada para que podamos considerarla como aleatoria pura.

La generación de números aleatorios sigue un proceso de relativa complejidad





La sentencia RANDOMIZE *sitúa la base (semilla) de la que se parte para la extracción de los números aleatorios.*

Ahora vamos a introducir algunas modificaciones en el programa, para que el ámbito de aplicación (desde cero hasta casi uno) se amplíe de acuerdo con nuestras necesidades:

```
10 REM - GENERACION CON LIMITE SUPERIOR
20 INPUT "Limite Superior:";B
30 FOR I=1 TO 44
40 PRINT RND*B,;
50 NEXT I
```

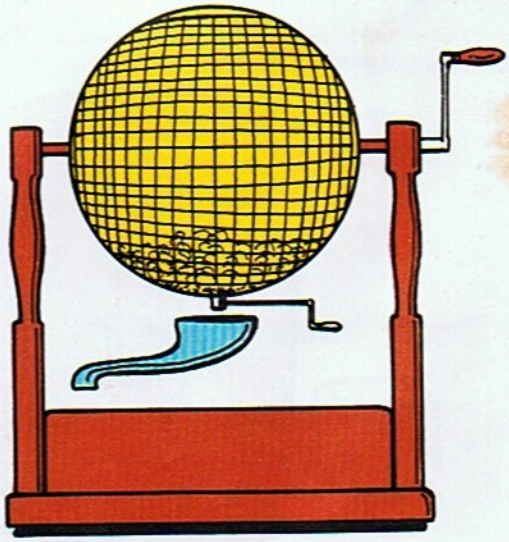
En primer lugar, en la línea 20, pedimos el límite superior de generación de números. En la línea 40, multiplicamos el número aleatorio generado, por el límite superior introducido por nosotros. De esta forma, obtenemos un número que oscila entre cero y el límite superior deseado, aunque sin alcanzarlo en ningún momento.
Como casos especiales, diremos que si introducimos cero como límite superior, todos los números generados serán cero; o si introducimos uno, nos encontraremos ante el mismo caso del programa ejemplo anterior, es decir, la generación aleatoria sin ningún tipo de conversión (intervalo 0 <= R < 1).
Si seleccionamos un número cualquiera, por ejemplo 10, encontraremos que las partes enteras de los valores obtenidos, oscilan entre 0 y 9. Este hecho puede darnos una pista de cómo obtener números naturales, a partir de las posibilidades de la función generadora de números aleatorios, y en combinación con una función que vimos anteriormente: **INT**, que nos permite convertir los números decimales en enteros.
Vamos a completar ahora el programa haciendo uso de la función **INT** *(INTeger)*, que acabamos de mencionar:

```
10 REM - GENERACION NATURAL CON LIMITE SUPERIOR
20 INPUT "Limite Superior:";B
30 FOR I=1 TO 44
40 PRINT INT(RND*B),;
50 NEXT I
```

Si seleccionamos once como límite superior, veremos que se obtiene una serie de números, que oscilan entre cero y el número indicado por nosotros como límite superior menos uno, es decir, de cero a diez. Más de una vez hemos llegado a pensar que algún profesor ponía las calificaciones por «lotería», y quizá sea este un claro ejemplo de que no andábamos demasiado descaminados.
Por último, podemos incluir un límite inferior a la serie de aleatorios. Puesto que ésta parte siempre de cero, vamos a sumar el número que deseemos como límite inferior, obteniendo la fórmula general para el cálculo de números naturales aleatorios entre límites:

```
PRINT A+INT(RND*B)
```

En el «bombo» de la generación de números aleatorios, entran 65536 bolas (semillas).



¡!

Para la generación de números aleatorios, se parte de un número base (semilla), el cual se somete a un complejo proceso de transformación.

*

Los números aleatorios generados por **RND** se encuentran en el intervalo cero a uno, pudiendo alcanzar el primer valor, pero nunca el último (0<=R<1).

*

Para utilizar una semilla determinada en la obtención de **RND**, se utiliza la sentencia **RANDOMIZE** seguida del número que se desea emplear.

Ejecutando **RANDOMIZE 0**, o simplemente **RANDOMIZE**, la semilla que se utiliza para los números aleatorios, se obtiene a partir del tiempo que la máquina lleva conectada.

Para la generación de números aleatorios se emplea la función **RND**, que carece de argumento.

Los números aleatorios generados por el Spectrum son decimales que nunca alcanzan el valor uno, aunque sí el número cero.

Siendo **A** el límite inferior y **B** la diferencia entre límite superior e inferior. Podemos hacer una prueba de esto con el siguiente programa:

```
10 REM - GENERACION NATURAL ENTRE LIMITES
20 INPUT "Limite Inferior:";A
30 INPUT "Limite Sup.-Inf.:";B
40 FOR I=1 TO 44
50 PRINT A+INT(RND*B),;
60 NEXT I
```

Conviene que no pasemos por alto la importancia de la fórmula que acabamos de ver, puesto que ella nos permitirá resolver los problemas planteados en la simulación de la mayoría de los juegos de azar: loterías, cartas, aventuras, etc...

La función BASIC encargada de la generación de números aleatorios es RND.

ALGUNAS COMPROBACIONES

Para los practicantes de la «duda metódica», podemos decir que no hay nada más fácil que comprobar la aleatoriedad de los resultados de la función **RND**. Para ello basta con la proposición de un par de programas de ejemplo, los cuales nos van a mostrar las posibilidades de obtener determinada puntuación con el lanzamiento de uno y dos dados, respectivamente.

En el primer caso, vamos a ver los resultados que se obtiene al lanzar cien veces consecutivas un dado, mostrando las ocurrencias de cada una de las posibilidades:

```
  10 REM - LANZAMIENTO DE UN DADO 100 VECES - J.M.LOPEZ MARTINEZ
  20 LET S1=0: LET S2=0: LET S3=0: LET S4=0: LET S5=0: LET S6=0
  30 PRINT TAB 2;
  40 FOR I=1 TO 30: PRINT (STR$ I)(LEN STR$ I);: NEXT I
  50 FOR I=1 TO 6: PRINT ''(' '+STR$ I)(LEN STR$ I TO ): NEXT I
  60 FOR I=1 TO 100
  70 LET R=1+INT (RND*6)
  80 IF R=1 THEN  LET S1=S1+1
  90 IF R=2 THEN  LET S2=S2+1
 100 IF R=3 THEN  LET S3=S3+1
 110 IF R=4 THEN  LET S4=S4+1
 120 IF R=5 THEN  LET S5=S5+1
 130 IF R=6 THEN  LET S6=S6+1
 140 IF S1 THEN  PRINT AT 2,S1+1;"3"
 150 IF S2 THEN  PRINT AT 5,S2+1;"3"
 160 IF S3 THEN  PRINT AT 8,S3+1;"3"
 170 IF S4 THEN  PRINT AT 11,S4+1;"3"
 180 IF S5 THEN  PRINT AT 14,S5+1;"3"
 190 IF S6 THEN  PRINT AT 17,S6+1;"3"
 200 NEXT I
```

Estudiemos rápidamente el programa: en la línea 20, se asigna el valor cero a los contadores de sucesos de las seis posibilidades. En las líneas 30 a 50, se construye la regla de la parte superior de la pantalla, para medir las ocurrencias, que indica la numeración desde 1 hasta 30. En la línea 60, se establece un bucle de 100 tiradas. En la línea 70, se simula la tirada del dado **R**. En las líneas 80 a 130, se suman la ocurrencias. En las líneas 140 a 190, se imprime, en la fila y columna correspondientes, la valoración de las ocurrencias y, por último, en la línea 200, se cierra el bucle de 100 tiradas.

La técnica para simular el diagrama de barras, que es como se conoce a este tipo de gráfico, es bien sencilla. Si nos fijamos un poco en el programa, veremos que, después de contabilizar las ocurrencias, se imprimen mediante sentencias **PRINT AT**, los caracteres que se obtienen por pulsación de **SHIFT** y **3**, con el cursor en el modo gráfico **G**.

Los números de fila (primer parámetro de **AT**) son

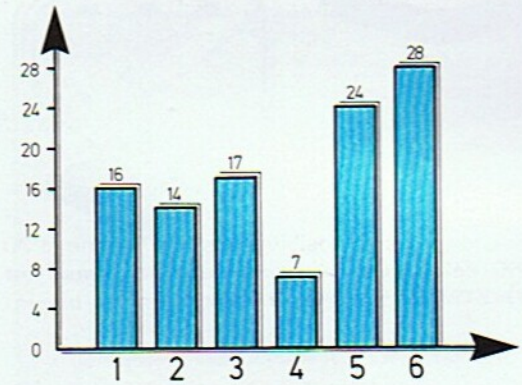
La función **INT** *colabora con* **RND** *en la simulación de ciertos sucesos aleatorios.*

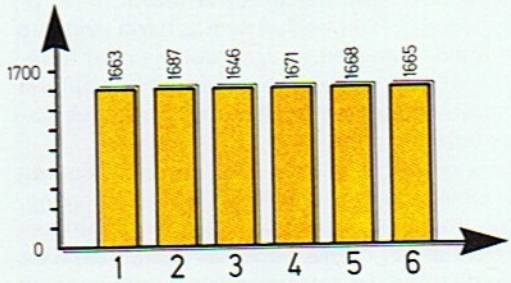
fijos, y corresponden a las líneas en que hemos situado los números del 1 al 6, en el margen izquierdo de la pantalla, para señalar qué suceso representa cada barra.

Los números de columna (segundo parámetro de **AT**), se calculan como el número de ocurrencias más uno. Dado que cada instrucción de este tipo la comenzamos con la comparación **IF S*n***, siendo **n** de 1 a 6, sólo se produce la impresión cuando **S*n*** es mayor que cero. De este modo, como sumamos uno, conseguimos que las impresiones se produzcan desde la columna tercera de la pantalla (**TAB 2**) hasta la columna 32 (**TAB 31**), si llega a alcanzarse esta posición, consiguiendo el efecto de una barra continua.

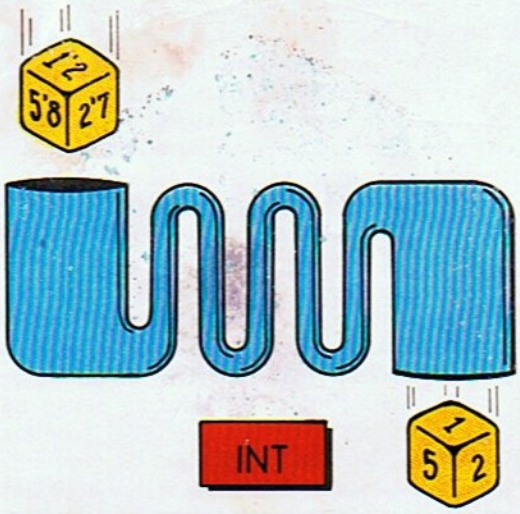
Observaremos que con cada nueva ejecución del programa por medio de la sentencia **RUN**, se obtienen proporciones diferentes de cada una de las posibilidades. Esto sucede porque aplicamos un número bajo de lanzamientos; puesto que, estadísticamente, es idéntica la posibilidad de salir cada una de las caras del dado hacia arriba, es de esperar que, con un número mucho mayor de lanzamientos, las longitudes de las barras se vayan igualando ante nuestros propios ojos.

Vamos a ver lo que sucedería elevando el número de ocurrencias hasta diez mil, aunque a base de armarnos de paciencia, porque el programa tardará en ejecutarse unos veinticinco minutos.





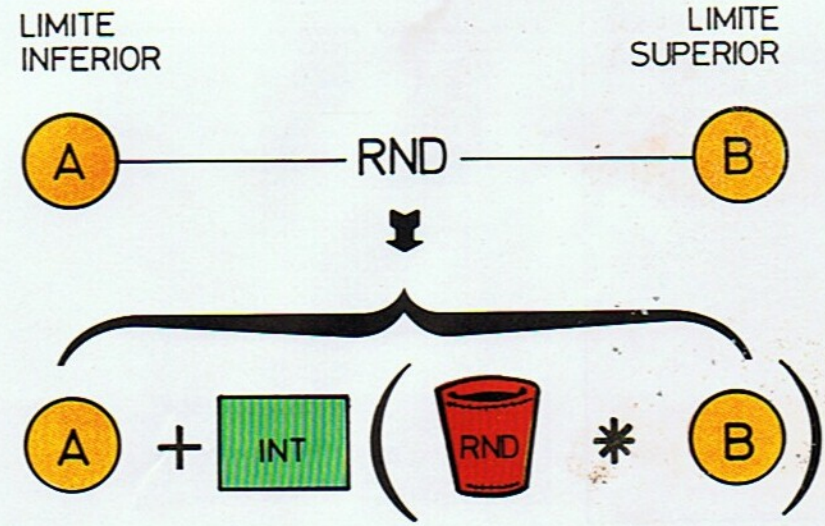
Gráficos obtenidos por el lanzamiento de un dado 100 y 10000 veces.



No obstante, nuestro afán investigador se verá recompensado al comprobar, de una forma experimental, que existen casi las mismas probabilidades para cada una de las ocurrencias (la longitud de las barras es similar), considerando un número alto de tiradas:

```
  10 REM - LANZAMIENTO DE UN DADO 10000 VECES - J.M.L
OPEZ MARTINEZ
  20 LET S1=0: LET S2=0: LET S3=0: LET S4=0: LET S5=0
: LET S6=0
  30 PRINT TAB 2;
  40 FOR I=1 TO 30: PRINT (STR$ I)(LEN STR$ I);: NEXT
 I
  50 FOR I=1 TO 6: PRINT ''(' '+STR$ I)(LEN STR$ I TO
): NEXT I
  60 FOR I=1 TO 10000
  70 LET R=1+INT (RND*6)
  80 IF R=1 THEN  LET S1=S1+1
  90 IF R=2 THEN  LET S2=S2+1
 100 IF R=3 THEN  LET S3=S3+1
 110 IF R=4 THEN  LET S4=S4+1
 120 IF R=5 THEN  LET S5=S5+1
 130 IF R=6 THEN  LET S6=S6+1
 140 IF S1>100 THEN  PRINT AT 2,S1/100+1;'3'
 150 IF S2>100 THEN  PRINT AT 5,S2/100+1;'3'
 160 IF S3>100 THEN  PRINT AT 8,S3/100+1;'3'
 170 IF S4>100 THEN  PRINT AT 11,S4/100+1;'3'
 180 IF S5>100 THEN  PRINT AT 14,S5/100+1;'3'
 190 IF S6>100 THEN  PRINT AT 17,S6/100+1;'3'
 200 NEXT I
```

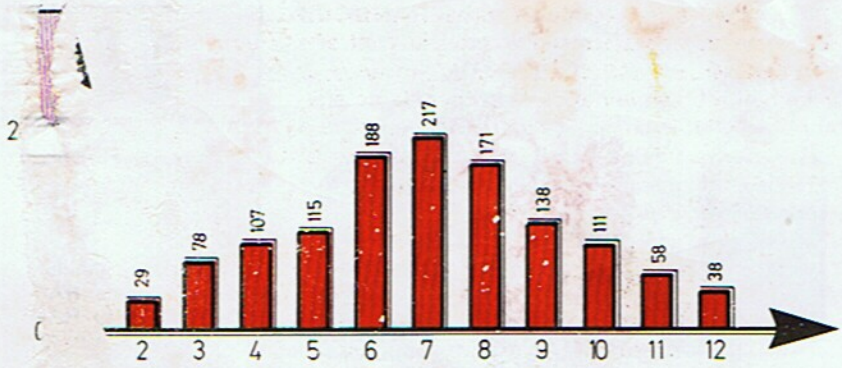
El algoritmo para la obtención de un natural entre límites es muy importante: R=A+INT (RND*B).



¡!

El empleo repetido y continuado de **RANDOMIZE 0** (o **RANDOMIZE**), en vez aleatorizar los resultados de **RND**, produce efectos desastrosos de repetición de resultados.

La forma de empleo de **RND** es similar a la de cualquier otra función BASIC.



La gráfica correspondiente a las ocurrencias en el lanzamiento de dos dados, adopta la forma conocida como «campana de Gauss».

Las modificaciones que hemos introducido en el programa no son muchas. En primer lugar, el número de lanzamientos es de diez mil, y por tanto se ha alterado el **FOR** de la línea 60. Por otra parte, el número de ocurrencias es dividido por cien antes de su presentación en la pantalla; al ser esta proporcional, cada una de las posiciones de la regla de la parte superior de la pantalla, representa cien ocurrencias en lugar de una.

Comenzamos ahora con el segundo de los experimentos propuestos: el caso del lanzamiento de dos dados. Esta vez, la tirada mínima es de dos y la máxima de doce, pero no son iguales las probabilidades de que aparezca cada una de ellas. Pasaremos primero a la práctica y después comentaremos los resultados:

¡!

Los números subrayados que se emplean en los listados de ejemplo, corresponden a los gráficos de las teclas; los subrayados doblemente, se obtienen mediante el gráfico con **CAPS SHIFT** (gráfico cambiado). Así por ejemplo, 3, corresponde al gráfico del 3 (▬); y 8 al gráfico con **CAPS SHIFT** de 8 (■).

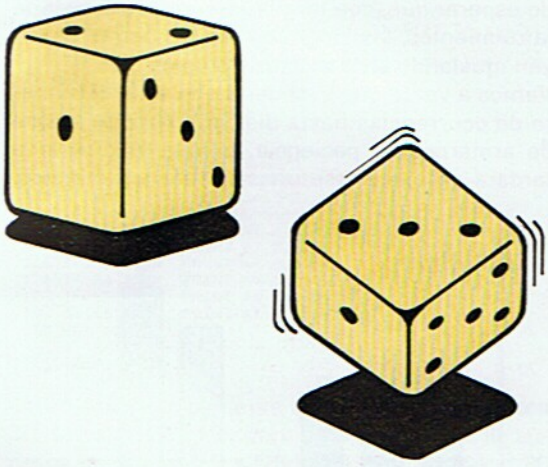
```
 10 REM - LANZAMIENTO DE DOS DADO 1250 VECES - J.M.L
OPEZ MARTINEZ
 20 LET S2=0: LET S3=0: LET S4=0: LET S5=0: LET S6=0
 30 LET S7=0: LET S8=0: LET S9=0: LET S10=0: LET S11
=0: LET S12=0
 40 PRINT TAB 2;
 50 FOR I=1 TO 30: PRINT (STR$ I)(LEN STR$ I);: NEXT
I
 60 FOR I=2 TO 12: PRINT '(' '+STR$ I)(LEN STR$ I TO
): NEXT I
 70 FOR I=1 TO 1250
 80 LET R=1+INT (RND*6): LET R=R+1+INT (RND*6)
 90 IF R=2 THEN  LET S2=S2+1
100 IF R=3 THEN  LET S3=S3+1
110 IF R=4 THEN  LET S4=S4+1
120 IF R=5 THEN  LET S5=S5+1
130 IF R=6 THEN  LET S6=S6+1
131 IF R=7 THEN  LET S7=S7+1
132 IF R=8 THEN  LET S8=S8+1
133 IF R=9 THEN  LET S9=S9+1
134 IF R=10 THEN  LET S10=S10+1
135 IF R=11 THEN  LET S11=S11+1
136 IF R=12 THEN  LET S12=S12+1
150 IF S2>20 THEN  PRINT AT 1,S2/10;'80'
160 IF S3>20 THEN  PRINT AT 3,S3/10;'80'
170 IF S4>20 THEN  PRINT AT 5,S4/10;'80'
180 IF S5>20 THEN  PRINT AT 7,S5/10;'80'
190 IF S6>20 THEN  PRINT AT 9,S6/10;'80'
200 IF S7>20 THEN  PRINT AT 11,S7/10;'80'
210 IF S8>20 THEN  PRINT AT 13,S8/10;'80'
220 IF S9>20 THEN  PRINT AT 15,S9/10;'80'
230 IF S10>20 THEN  PRINT AT 17,S10/10;'80'
240 IF S11>20 THEN  PRINT AT 19,S11/10;'80'
250 IF S12>20 THEN  PRINT AT 21,S12/10;'80'
310 NEXT I
```

Puesto que la técnica de programación es similar a la de los programas ejemplos anteriores, no merece la pena que recalquemos nada especialmente sobre su estructura. Sólo diremos que cada posición de la escala equivale a diez ocurrencias, y que a la cabeza de las barras hemos añadido una marca del valor máximo.

Sí es interesante observar que, esta vez, se aprecian sensibles diferencias entre las ocurrencias de cada uno de los sucesos, sobre todo entre las barras centrales y las situadas en los extremos superior e inferior, respectivamente.

Esto es debido a que las probabilidades no son realmente las mismas. De hecho, existe una sola posibilidad de obtener en una tirada tanto el 2 como el 12, que se corresponde con el 1 y el 6 de los dos dados, respectivamente.

En cambio, la probabilidad de obtener una puntuación de 3 o de 11, es exactamente el doble, ya que existen dos combinaciones válidas para cada caso: para la puntuación primera, el 1 en el primer dado y el 2 en el segundo, o el 2 en el primer dado y el 1 en el segundo. De forma análoga, se explica la doble probabilidad de obtener la puntuación de 11



RANDOMIZE *seguido de un número, se utiliza para desaleatorizar los resultados de* **RND.**

De esta manera progresiva, van aumentando las probabilidades hasta alcanzar la mayor, que corresponde al 7. Este hecho, se nos haría evidente efectuando, como en el caso del ejemplo anterior, un alto número de tiradas, pero resultaría realmente tedioso por su lentitud, por lo que haremos un «acto de fe».

Si observamos el gráfico, nos daremos cuenta que adopta una forma parecida a una campana, más abultada en el centro, y disminuida uniformemente en su altura hacia los extremos. Este tipo de distribución se conoce como «campana de Gauss», en honor al prestigioso matemático que investigó sobre las leyes de la probabilidad.

RANDOMIZE

La sentencia **RANDOMIZE** (en abreviatura **RAND**) tiene por objetivo hacer comenzar la generación de aleatorios de la función **RND**, en un punto preciso de la secuencia de números semilla, es decir, sentar el origen para esta función. Si se ejecuta una sentencia **RANDOMIZE 1** seguida de varios **RND**, se obtiene una serie de valores aleatorios; pero si se vuelve a ejecutar de nuevo **RANDOMIZE 1**, los valores calculados a continuación por dicha función, serán idénticos a los calculados anteriormente, es decir, nada aleatorios:

```
10 REM - REPETICION DE SECUENCIAS
20 RANDOMIZE 1
30 FOR I=0 TO 21:PRINT AT I,0;RND:NEXT I
40 RANDOMIZE 1
50 FOR I=0 TO 21:PRINT AT I,16;RND:
NEXT I
```

Existe una segunda forma de utilizar **RANDOMIZE**, esta vez para conseguir todo lo contrario. **RANDOMIZE** o **RANDOMIZE 0**, provocan un cambio en el elemento de la secuencia de aleatorios a utilizar por la función **RND**, en base al reloj interno del Spectrum. En pocas palabras, sitúan la semilla según el tiempo transcurrido desde la conexión de la máquina.

Esto es muy útil cuando se trata de obtener un número realmente aleatorio, pero desastroso cuando un programa pasa repetidas veces, y en corto espacio de tiempo, por una instrucción que contenga un **RANDOMIZE** de este tipo, ya que su efecto es una repetición grande de los resultados.

Una muestra clara de la aplicación de **RANDOMIZE**, la tenemos en el programa MASTER CUBOS, del que tuvimos oportunidad de disfrutar hace poco. En la línea 1290, que encabeza la presentación del juego, se ejecuta una instrucción **RANDOMIZE 2**; a partir de allí, los colores para los cubos que formarán en la pantalla la palabra gigante CUB, se obtienen mediante la función **RND**; a ello se debe que estos sean siempre del mismo color. Si no hubiéramos empleado este sistema, nos habríamos visto obligados a malgastar memoria y variables para almacenar los colores correspondientes a los cubos.

Al tipo de sucesos que dependen del azar, se les conoce como SUCESOS ALEATORIOS o ESTOCASTICOS.

Dos son las sentencias que controlan el proceso de generación de números aleatorios: **RANDOMIZE** (abreviado en el teclado a **RAND**) y **RND**.

ALGUNOS EJERCICIOS

Con lo dicho hasta el momento, tenemos la base suficiente para obtener y manejar la generación de números aleatorios. Así pues, vamos a hacer algunos programas a modo de ejercicio:

1. Extracción de una carta de la baraja española (40 cartas).

```
10 REM BARAJA ESPAÑOLA
20 REM R=NUMERO ALEATORIO
30 REM C=CARTA
40 REM P=PALO
50 LET R=INT (RND*40)
60 LET P=INT (R/10)
70 LET C=R-10*P
80 IF C=7 THEN  LET C$='SOTA': GO TO 130
90 IF C=8 THEN  LET C$='CABALLO': GO TO 130
100 IF C=9 THEN  LET C$='REY': GO TO 130
110 IF C=0 THEN  LET C$='AS': GO TO 130
120 LET C$=STR$ (C+1)
130 IF P=0 THEN  LET P$='OROS': GO TO 160
140 IF P=1 THEN  LET P$='COPAS': GO TO 160
150 IF P=2 THEN  LET P$='ESPADAS': GO TO 160
160 IF P=3 THEN  LET P$='BASTOS'
170 PRINT C$;' DE ';P$
```

2. Lanzamiento de dos dados (dados de 6 caras).

```
10 REM LANZAMIENTO DE DOS DADOS
20 REM D1=DADO 1
30 REM D2=DADO 2
40 LET D1=1+INT (RND*6)
50 LET D2=1+INT (RND*6)
60 PRINT 'DADO 1= ';D1,'DADO 2= ';D2
```

3. Moneda arrojada al aire (2 caras).

```
10 REM CARA O CRUZ
20 IF RND<.5 THEN  PRINT 'CARA'
30 PRINT 'CRUZ'
```

4. Extracción de una carta de la baraja francesa (54 cartas).

```
10 REM BARAJA FRANCESA
20 REM R=NUMERO ALEATORIO
30 REM C=CARTA
40 REM P=PALO
50 LET R=INT (RND*54)
60 IF R>51 THEN  PRINT 'COMODIN': GO TO 10000
70 LET P=INT (R/13)
80 LET C=R-13*P
90 IF C=10 THEN  LET C$='JACK': GO TO 140
100 IF C=11 THEN  LET C$='REINA': GO TO 140
110 IF C=12 THEN  LET C$='REY': GO TO 140
120 IF C=0 THEN  LET C$='AS': GO TO 140
130 LET C$=STR$ (C+1)
140 IF P=0 THEN  LET P$='CORAZONES': GO TO 180
150 IF P=1 THEN  LET P$='DIAMANTES': GO TO 180
160 IF P=2 THEN  LET P$='PICAS': GO TO 180
170 IF P=3 THEN  LET P$='TREBOLES'
180 PRINT C$;' DE ';P$
```

5. Estadística de las notas asignadas al azar (de 0 a 10).

```
10 REM ESTADISTICA DE NOTAS AL AZAR
20 REM N=NOTA
30 LET N=INT (RND*11)
40 PRINT 'NOTA= ';N
50 GO TO 30
```

6. Extracción de bolas de un bombo de bingo (90 bolas).

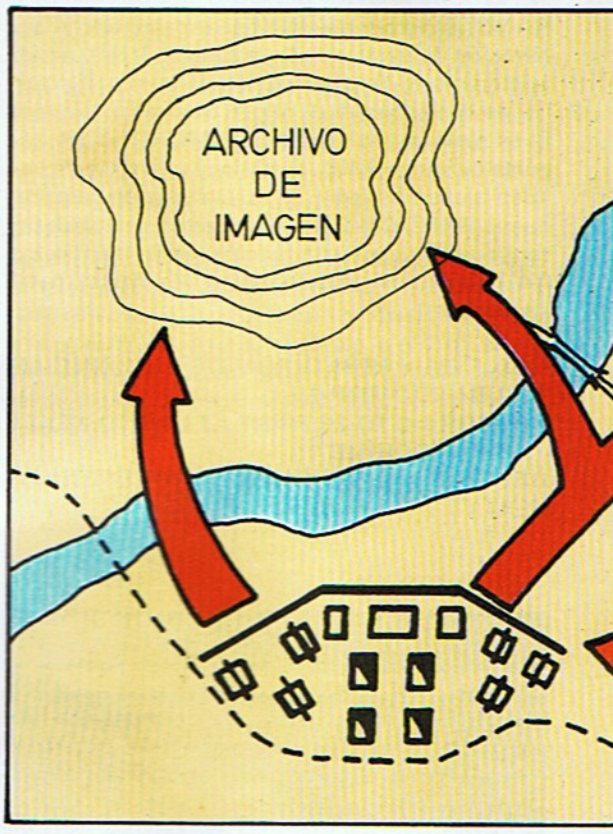
```
10 REM BINGO
20 REM B=BOLA EXTRAIDA
30 LET B=1+INT (RND*90)
40 PRINT 'BOLA: ';B
```

A TODA PANTALLA

EN el capítulo anterior, adquirimos los conocimientos necesarios sobre la pantalla del Spectrum, como para permitirnos en las páginas siguientes discutir una serie de técnicas encaminadas al dinamismo de pantallas. Fundamentalmente, nuestra acción se concentrará sobre dos puntos:

* El archivo de imagen.
* El área de atributos.

En una primera fase, nos dedicaremos al tratamiento conjunto de ambas zonas, para más adelante pasar a técnicas que afectan a cada una de ellas separadamente. Nuestro primer movimiento se refiere concretamente a la carga de pantallas previamente almacenadas, y fundamentalmente trata el aspecto de la dinámica inherente a la misma, proponiendo varios sistemas para alterar el proceso normal de carga en el Spectrum. En segundo lugar, concentraremos todas nuestras fuerzas sobre el área de atributos, estudiando una interesante subrutina en código máquina, con la cual poder obtener espectaculares efectos. Por último, nos adentraremos en el archivo de imagen, donde estudiaremos la más pura animación de pantallas, aunque lamentablemente las características de ésta reservan su disfrute a aquellos que poseamos el modelo de 48 K o el Plus, que en lo referente a la memoria, tanto monta, monta tanto.



BITS

Una alternativa al comando **NEW**, la proporciona **RANDOMIZE USR 4572**. Esta llamada a la R.O.M., tendrá el mismo efecto de borrado de la memoria, hasta el límite fijado por RAMTOP. Pero a diferencia de **NEW**, la pantalla no cambiará al negro durante unos instantes.

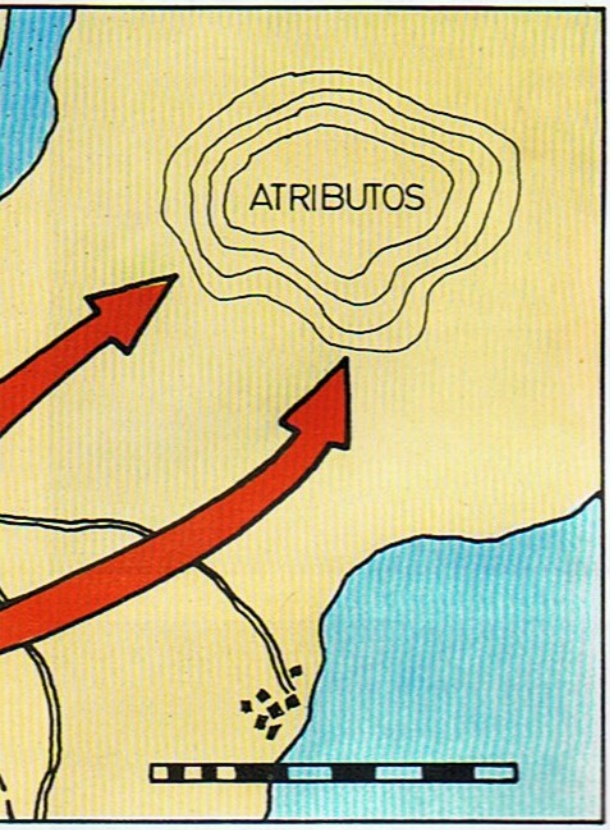
CARGA DE PANTALLAS

Una de las cosas que más llaman la atención al ver cargar un programa en el Spectrum, es la peculiar manera de construir las pantallas que, como ya vimos anteriormente, se debe a la organización un tanto caótica de su archivo de imagen.

El gran problema que se aprecia inmediatamente es que, lógicamente, se carga antes el archivo de imagen que su área de atributos; a causa de ello, el primer aspecto que presenta la pantalla al cargarse, antes de su finalización, no suele ser muy estético. Sería bueno disponer de un sistema que no permitiera ver la pantalla hasta que no se hubieran asignado los colores, evitando así primeras impresiones, generalmente bastante feas. Efectivamente, disponemos de este sistema, y además es aplicable de una manera bien sencilla: sólo hemos de asignar un mismo color de **INK** (tinta) y **PAPER** (papel) antes de la carga de la pantalla. Gracias a este pequeño truco, la pantalla no aparecerá hasta que se inicie la carga del área de atributos. Podemos probar la carga de una pantalla mediante el siguiente programa:

```
10 REM PANTALLA INVISIBLE - C. DE LA OSSA & F. LOPEZ MARTINEZ
20 BORDER 7: PAPER 7: INK 7: BRIGHT 0: FLASH 0: CLS
30 LOAD ""SCREEN$
```

Pero quizás los amantes de la rapidez prefieran algo más efectivo, más rápido aún; para este se-



Para dinamizar pantallas en conjunto, concentraremos nuestra acción en dos puntos: el archivo de imagen y el área de atributos.

gundo truco, necesitaremos el auxilio del código máquina. Nos basaremos en una instrucción LDIR (instrucción de ensamblador, pero no de BASIC), que permite la transferencia de bloques de memoria; de esta forma, podremos cargar la pantalla en una zona de memoria, que no sea la suya propia, y luego transferirla a la zona de memoria de pantalla de manera casi instantánea. El siguiente programa realiza esta función:

```
10 REM PANTALLA INSTANTANEA - C. DE
LA OSSA & F. LOPEZ MARTINEZ
20 CLEAR 25687
30 FOR I=0 TO 11
40 READ A: POKE USR "A"+I,A
50 NEXT I
60 DATA 33,88,100,17,0,64,1,0,27,237,
176,201
70 LOAD ""CODE 25688
80 RANDOMIZE USR USR "A"
```

Para aquellos que hagamos nuestros pinitos en el código máquina, añadiremos el ensamblador de esta rutina, que podríamos denominar de TRANSFERENCIA DE PANTALLAS:

SUBRUTINA DE TRANSFERENCIA

ETIQUETA	OBJETO	FUENTE	COMENTARIO
TRSF	33 88 100	LD HL,25688	Carga HL con la dirección de origen.
	17 0 64	LD DE,16384	Carga DE con la dirección de destino.
	1 0 27	LD BC,6912	Carga BC con la longitud del bloque a transferir.
	237 176	LDIR	Transfiere el bloque de memoria especificado.
	201	RET	Retorna al BASIC.

Por otra parte, la utilidad de esta rutina es doble, y no sólo se limita a la representación rápida de pantallas; veamos el porqué de esta afirmación. El tiempo que el Spectrum emplea para la carga de una pantalla no es precisamente poco (más de 36 segundos), y resulta un tanto frustrante que tanto esfuerzo desaparezca tras un brevísimo esplendor. Si observamos cuidadosamente el sistema de carga que hemos estudiado unas líneas más arriba, comprobaremos que la pantalla sigue aún situada en el lugar en que fue cargada, puesto que la rutina en código máquina simplemente la copia, pero no la deteriora ni cambia de lugar; por tanto, esta subrutina puede ser también empleada para restituir la imagen inicial en la pantalla siempre que lo consideremos oportuno, por ejemplo, en el cambio de jugadores en un juego de acción *(arcade)* o tras el mensaje de fin de juego.

Así pues, esta rutina que hemos visto nos puede ser de gran utilidad, ya que más que realizar una transferencia de pantallas, efectúa una copia o reproducción. Otra de sus cualidades es la de ser reubicable, lo cual significa que cualquiera que

stema de carga de alias en el Spectrum, te ver el archivo de antes de la n de color.

BITS

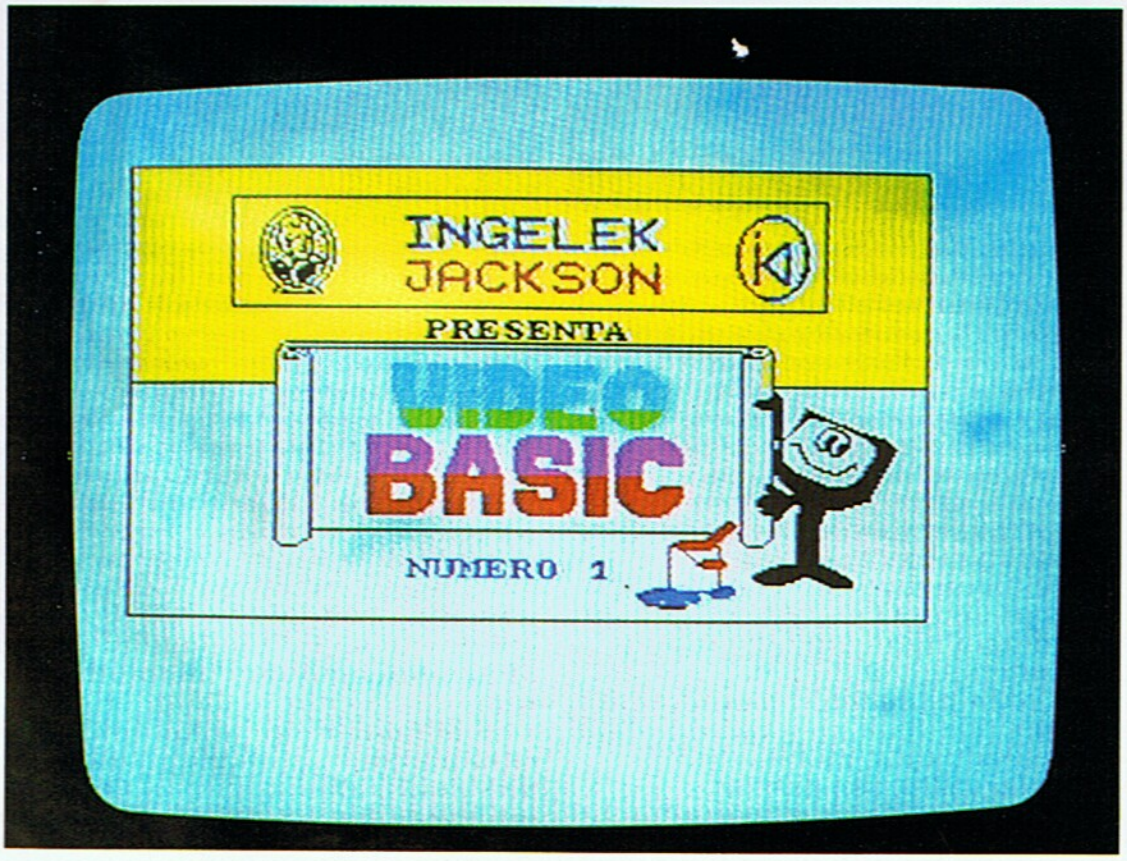
Las últimas hojas del manual de nuestro Spectrum, incluyen una lista de los posibles informes que el ordenador puede visualizar en la parte inferior de la pantalla, durante la ejecución de un programa. Probemos el efecto de las siguientes líneas:

```
10 POKE 23296,207
20 FOR N=0 TO 26
30 POKE 23297,N
40 PRINT USR 23296
50 NEXT N
```

Cada vez que obtengas un mensaje de error, con **GO TO 50** y **ENTER**, aparecerá uno nuevo en la parte inferior de la pantalla, hasta completar el programa con el conocido **0 OK**. Después de esto, ya no podremos decir que existe algún error que nunca hemos visto.

*

Cuando una estructura de bucle incluye a otro o varios se denomina «anidamiento».

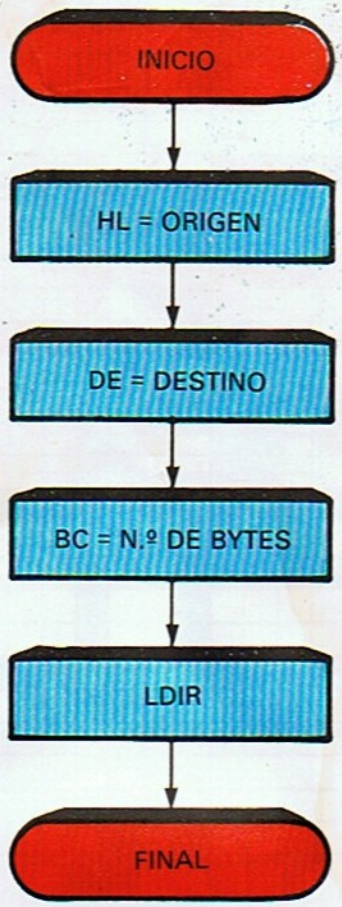


Mediante un truco bien simple, podemos conseguir que la pantalla no se reconozca hasta que no se haya cargado el área de atributos.

sea el lugar de la memoria donde se encuentre funcionará a la perfección. En nuestro caso, pensando en que el programa de ejemplo pudiera ser ejecutado en un modelo de 16 K o 48 K indistintamente, se ha empleado el área de gráficos definidos, concretamente a partir de la A, para su almacenamiento; sin embargo, el mismo efecto habríamos obtenido cargándola, por ejemplo, a partir de 25000 (decimal), y alterando por supuesto la llamada correspondiente de la línea 80 **(80 RANDOMIZE USR 25000)**.

Cuando hablemos de la animación del archivo de imagen, tendremos ocasión de comentar más ampliamente, y aprender a utilizar en nuestros propios programas, esta interesante subrutina; sin embargo, sobre este último sistema de carga de pantallas que se apoya en ella, hemos de añadir que ocupa una gran cantidad de memoria: principalmente los casi 7 K (6912 bytes) de la pantalla almacenada (los 12 bytes de subrutina son insignificantes), por tanto, es muy probable que los usuarios del modelo de 16 K no podamos permitirnos este despilfarro de memoria.

```
  10 REM RUN-SOFT - C.DE LA OSSA - F.LOPEZ MARTINEZ
  20 BORDER 0: PAPER 0: INK 2: CLS
  30 PRINT AT 3,1; FLASH 1;"88888"; FLASH 0;"    "; FL
ASH 1;"8"; FLASH 0;"    "; FLASH 1;"8"; FLASH 0;"  ";
 FLASH 1;"8"; FLASH 0;"    "; FLASH 1;"8"
  40 PRINT AT 4,1; FLASH 1;"8"; FLASH 0;"    "; FLASH
 1;"8"; FLASH 0;"  "; FLASH 1;"8"; FLASH 0;"    "; FL
ASH 1;"8"; FLASH 0;"  "; FLASH 1;"88"; FLASH 0;"  ";
 FLASH 1;"8"
  50 PRINT AT 5,1; FLASH 1;"8"; FLASH 0;"    "; FLASH
 1;"8"; FLASH 0;"  "; FLASH 1;"8"; FLASH 0;"  "; FL
ASH 1;"8"; FLASH 0;"  "; FLASH 1;"8"; FLASH 0;" "; FL
ASH 1;"8"; FLASH 0;"  "; FLASH 1;"8"
  60 PRINT AT 6,1; FLASH 1;"88888"; FLASH 0;"    "; FL
ASH 1;"8"; FLASH 0;"    "; FLASH 1;"8"; FLASH 0;" ";
 FLASH 1;"8"; FLASH 0;"  "; FLASH 1;"8"; FLASH 0;" ";
 FLASH 1;"8"; FLASH 0;"  "; FLASH 1;"888888"
  70 PRINT AT 7,1; FLASH 1;"8"; FLASH 0;"    "; FLASH
 1;"8"; FLASH 0;"    "; FLASH 1;"8"; FLASH 0;"    "; FL
ASH 1;"8"; FLASH 0;"  "; FLASH 1;"8"; FLASH 0;"  ";
FLASH 1;"88"
  80 PRINT AT 8,1; FLASH 1;"8"; FLASH 0;"    "; FLASH
 1;"8"; FLASH 0;"    "; FLASH 1;"8888"; FLASH 0;"  ";
 FLASH 1;"8"; FLASH 0;"    "; FLASH 1;"8"
  90 PAPER 0: INK 5
 100 PRINT AT 13,2; FLASH 1;"    "; FLASH 0;"    "; F
LASH 1;"    "; FLASH 0;"  "; FLASH 1;"    "; FLASH
 0;" "; FLASH 1;"    "
 110 PRINT AT 14,1; FLASH 1;" "; FLASH 0;"    "; F
LASH 1;" "; FLASH 0;"    "; FLASH 1;" "; FLASH 0;" "
; FLASH 1;" "; FLASH 0;"    "; FLASH 1;" "
 120 PRINT AT 15,2; FLASH 1;"    "; FLASH 0;"    "; FL
ASH 1;" "; FLASH 0;"    "; FLASH 1;" "; FLASH 0;" ";
 FLASH 1;"    "; FLASH 0;"    "; FLASH 1;" "
 130 PRINT AT 16,6; FLASH 1;" "; FLASH 0;"  "; FLASH
1;" "; FLASH 0;"    "; FLASH 1;" "; FLASH 0;"  "; FLA
SH 1;" "; FLASH 0;"    "; FLASH 1;" "
 140 PRINT AT 17,6; FLASH 1;" "; FLASH 0;"  "; FLASH
1;" "; FLASH 0;"    "; FLASH 1;" "; FLASH 0;"  "; FLA
SH 1;" "; FLASH 0;"    "; FLASH 1;" "
 150 PRINT AT 18,2; FLASH 1;"    "; FLASH 0;"    "; F
LASH 1;"    "; FLASH 0;"    "; FLASH 1;" "; FLASH 0;"
    "; FLASH 1;" "
 160 FOR I=0 TO 56
 170 READ A,B
 180 BEEP A,B
 190 NEXT I
 200 DATA .5,5,.5,4,.5,2,.5,0,.2,9,.08,9,.08,9,.2,9
 210 DATA .25,0,.7,9,.3,-59,.5,5,.5,4,.5,5,.5,2,.2,10
,.08,10
 220 DATA .08,10,.2,10,.25,2,.7,10,.3,-59,.35,7,.1,6,
.35,7,.1,6
 230 DATA .35,7,.1,6,.20,7,.2,10,.2,9,.08,9,.08,7,.1,
9,.1,-59
 240 DATA .1,10,.1,-59,.2,12,.2,-1,.2,0,.2,1,.2,2,.08
,2,.08,1
 250 DATA .2,2,.25,5,.2,4,.1,4,.1,3,.1,4,.1,-59,.2,7,
.3,5
 260 DATA .2,-59,.3,0,.2,-59,.3,5
 270 PAUSE 250
 280 RUN
```



Organigrama de la subrutina de transferencia.

INTERMITENCIA

Para terminar con esta primera parte de la animación a toda pantalla, entraremos de lleno en el tema que nos ocupa: el dinamismo. El ordenador no es capaz de ejecutar dos programas a un mismo tiempo, sin embargo, asistiremos con asombro a la interpretación de una musiquilla mientras la pantalla no deja de «bailar». ¿Magia? No, un simple truco basado en el modo **FLASH**; como sabemos, el **FLASH** o intermitencia es una tarea llevada a cabo de forma automática por el *hardware* del ordenador, y por tanto siempre puede actuar, aunque se estén realizando otras funciones, como por ejemplo, la carga de programas. Pasemos a la práctica.

El truco consiste en la creación de dos mensajes gigantes en la pantalla, que escribiremos me-

RUNEAR: Ejecutar un programa mediante el comando **RUN**.

*

POKEAR: Ejecutar una sentencia **POKE**.

*

FORMATEAR: Someter un cartucho a una operación **FORMAT**.

*

LEER: Cargar un programa o datos desde la cinta. Obtener algún valor mediante una función.

*

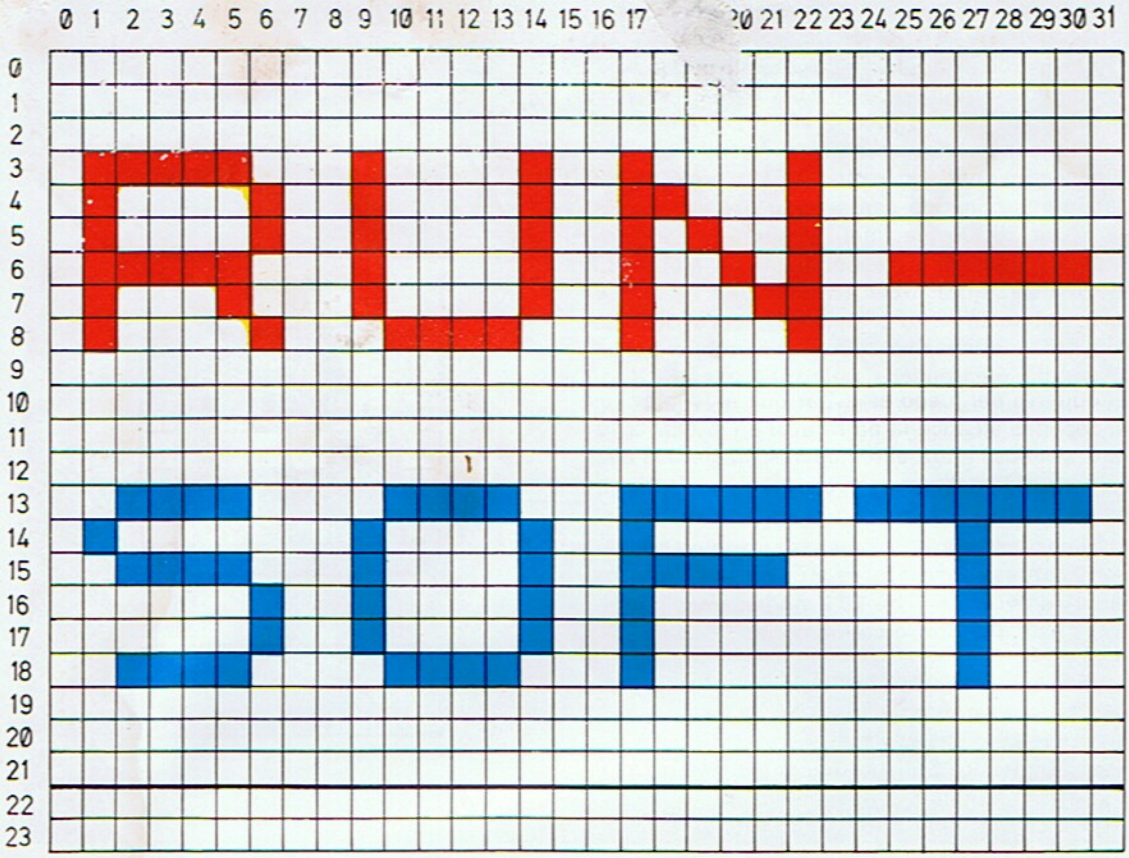
MERGEAR: Mezclar un programa con el de la memoria mediante el comando **MERGE**.

*

PLOTEAR: Ejecutar una sentencia **PLOT**.

*

PRINTEAR: Escribir en la pantalla mediante la sentencia **PRINT**.



diante cuadrados macizos de la dimensión de un carácter, los cuales podremos conseguirlos a partir del gráfico cambiado del 8; es decir, **CAPS SHIFT + 8**, con el cursor en modo **GRAPHICS (CAPS SHIFT + 9)**. La posición en que se escriban los caracteres, no es en este caso lo más importante, sino la sincronización del modo **FLASH** y los colores correspondientes, para que se produzca un efecto de movimiento por intermitencia. Para acabar de perfilar la impresión óptica, haremos que el sistema de «puesta en escena» de los mensajes, sea contrario en cuanto a códigos de tinta y fondo, gracias a lo cual, cuando un mensaje se presente apagado, el otro aparecerá encendido y viceversa. Lógicamente, para conseguir el efecto de desaparición del mensaje, deberemos utilizar el mismo color de fondo para éste que para el resto de la pantalla.

Llevando aún más lejos este efecto óptico, podemos entremezclar dos mensajes, haciéndolos aparecer en una misma zona de la pantalla, aunque en este caso la coordinación de colores será más complicada, puesto que el color de «apagado» ya no corresponderá siempre con el del fondo de la pantalla, sino con el de «encendido» del otro mensaje. Veamos un ejemplo práctico con dos únicos caracteres, que sirven de nombre de pila a nuestro aparato: ZX.

```
10 REM Z-X - C.DE LA OSSA - F.LOPEZ MARTINEZ
20 BORDER 0: PAPER 0: FLASH 0: BRIGHT 0: INK 0: CLS

30 INK 2
40 PRINT AT 8,13; FLASH 1; PAPER 6;"8"; PAPER 0;"88
88"; PAPER 6;"8"
50 PRINT AT 9,14; FLASH 1; PAPER 6; INK 0;"8"; PAPE
R 0;"  "; PAPER 6; INK 2;"8"
60 PRINT AT 10,15; FLASH 1; PAPER 6; INK 0;"8"; PAP
ER 6; INK 2;"8"
70 PRINT AT 11,15; FLASH 1; PAPER 6; INK 2;"8"; INK
 0;"8"
80 PRINT AT 12,14; FLASH 1; PAPER 6;"8"; PAPER 0; I
NK 0;"  "; PAPER 6; INK 0;"8"
90 PRINT AT 13,13; FLASH 1; PAPER 6;"8"; PAPER 0;"8
888"; PAPER 6;"8"
```

Ha llegado el momento de la despedida, pero eso sí, temporal. En nuestro próximo capítulo estudiaremos el resto de técnicas de animación a pantalla completa. Aprovechamos la ocasión para advertir, que para uno de los programas de demostración, va a ser necesario el programa de gráficos en tres dimensiones, aparecido en el segundo capítulo de nuestra sección dedicada a la programación.

En todo caso, sería conveniente que practicásemos con el sistema de animación por **FLASH** que hemos aprendido, puesto que, aunque aparentemente es fácil conseguir el efecto deseado, en la mayoría de los casos llegamos a resultados frustrantes; máxime si se utiliza el segundo tipo de animación, en el cual las letras se entrecruzan, haciendo de la pantalla un misterio difícil de desentrañar.

¿?

SALVAR: Grabar un programa en cinta, disco o cualquier otro soporte, mediante el comando **SAVE**.

*

BIT: Unidad mínima de información. Se representa por un uno o un cero binario.

*

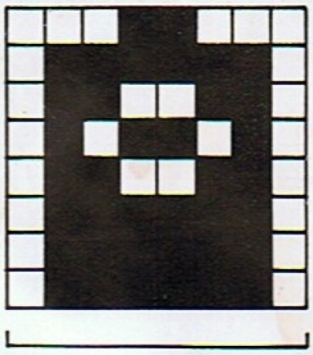
BYTE: Agrupación de ocho bits. La información que puede contener un byte va desde 0 hasta 255.

¡ARAÑAS!

ESTE programa nos brinda la oportunidad de convertirnos en un spray super eficaz con el que defendernos del despiadado ataque de unos arácnidos.
El objetivo del juego es evitar que alguno de los artrópodos alcance la barra azul en la cual estamos situados. Para ello, nos desplazaremos a izquierda y derecha, pulsando las teclas Z y X respectivamente, y lanzaremos nuestra mortífera carga de DDT por medio de la tecla SPACE. Cuanto mayor sea el número de puntos conseguido, más rápidamente se descolgarán las arañas por la pantalla. Si al final del juego hemos logrado batir el récord, nuestro Spectrum nos lo hará saber mediante una serie de notas y palpitaciones en el marcador (sin duda debidas a la emoción del momento).
Aunque nuestro SPRAY tiene la suficiente carga como para eliminar miles y miles de arañas, debido a la gran rapidez de los arácnidos estamos irremisiblemente condenados a la desaparición; sin embargo, la competencia se puede establecer entre nosotros y los otros jugadores, o bien con nosotros mismos, intentando batir los récords de permanencia establecidos.

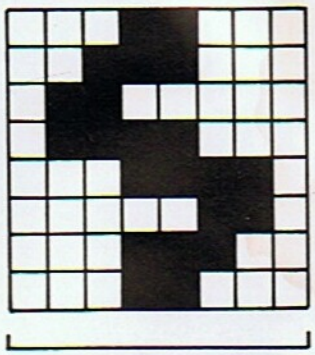
VARIOS NIVELES

Este programa nos da pie a introducir un concepto importante en la confección de juegos de acción: no hay nada más desesperante que un juego en el que por muy experimentado que se sea, nunca se consiga alcanzar un mínimo número de puntos; para evitar esto, hemos de escoger un nivel intermedio para la dificultad del juego, que no sea ni tediosamente sencillo, ni increiblemente difícil.
En el programa ARACNIDOS, el control de esta dificultad es gestionado por la línea 210, mediante el siguiente algoritmo: **RND*10000 >P+7000;** cuando esta condición se cumple, no



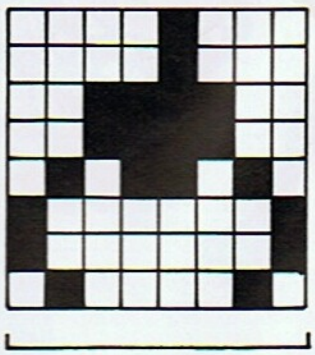
SPRAY

El spray se consigue gracias al gráfico definido de la A.



MATA-ARAÑAS

El mortífero chorro de insecticida, se representa mediante el gráfico definido de la letra B.



ARAÑA

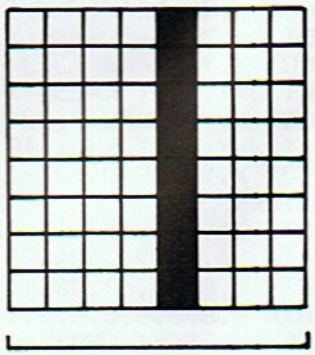
Los arácnidos corresponden a los gráficos definidos de la C.

se ejecuta el grupo de instrucciones que efectúa el avance de las arañas.
Observando el algoritmo, se aprecia que en el mejor de los casos, al comenzar el juego, la probabilidad de avance del arácnido es de aproximadamente el 30 por 100, y conforme vamos adentrándonos en el juego, esta probabilidad aumenta hasta alcanzar el 100 por 100, lo cual sucede al rebasar los 3000 puntos.

CARACTERES SUBRAYADOS

Como es habitual, los caracteres que aparecen subrayados en el listado, deben sustituirse por los gráficos definidos de las teclas correspondientes; así por ejemplo, una A subrayada, corresponde a un carácter gráfico de la A, que debe ser introducido con el ordenador en modo **GRAPHICS (CAPS SHIFT + 9),** y pulsando a continuación la tecla A, sin olvidar desconectar a continuación el modo GRAFICO con **CAPS SHIFT + 9** o, por tratarse de la desconexión, simplemente con **9**.
La adopción del programa se efectuará por el método habitual, puesto que este no incluye código máquina: **SAVE "ARACNIDOS",** o bien, **SAVE "ARACNIDOS" LINE 10,** para que al finalizar la carga, el programa se ejecute automáticamente desde la línea 10.

Con el gráfico definido de la D, se representan los hilos por los que se descuelgan las arañas.



HILO

¡!

Los caracteres subrayados que aparecen en el listado del programa, corresponden a los gráficos definidos, por lo que a la hora de introducir el programa, pondremos el cursor en modo gráficos, pulsando posteriormente la letra subrayada.

*

El algoritmo que controla la dificultad de juego es: **RND*10000 >P+ 7000.**

*

El movimiento del spray se efectúa mediante la pulsación de las teclas Z (izquierda), X (derecha) y SPACE (fuego).

*

Para grabar el programa ejecutaremos el comando: **SAVE "ARACNIDOS" LINE 10.**

```
 10 BORDER 5: PAPER 5: INK 6: BRIGHT 0: CLS
 20 LET R=0: POKE 23658,8
 30 FOR I=0 TO 31
 40 READ J
 50 POKE USR "A"+I,J
 60 NEXT I
 70 BORDER 5: PAPER 5: INK 6: BRIGHT 0: CLS
 80 FOR I=0 TO 30: GO SUB 300: NEXT I
 90 INK 2: PAPER 5: PRINT AT 4,7; BRIGHT 1;"
          ": FOR I=5 TO 16: PRINT AT I,7; BRIGHT 1;"
";AT I,24; BRIGHT 1;" ": NEXT I: PRINT AT 17,7; BRIGH
T 1;"                    "
 100 BRIGHT 0: PRINT AT 1,1; PAPER 1; INK 9;"PUNTOS:0
000";AT 1,20;"RECORD:";FN P$(R);AT 20,1; PAPER 1; INK
 7;"Z=IZDA. X=DECHA. ´SPACE´=SPRAY": PAPER 6: BRIGHT
1
 110 PRINT AT 5,8; PAPER 2;TAB 24
 120 FOR I=6 TO 15: PRINT AT I,8;TAB 24: NEXT I
 130 PRINT AT 16,8; PAPER 1;TAB 24
 140 LET X=16: DIM A(16): LET P=0: PRINT AT 16,16; PA
PER 1; INK 7;"A"
 150 LET X$=INKEY$
 160 IF X$="Z" AND X>8 THEN  LET X1=X-1: GO TO 200
 170 IF X$="X" AND X<23 THEN  LET X1=X+1: GO TO 200
 180 IF X$=" " THEN  FOR I=14 TO 6 STEP -1: PRINT AT
I,X; INK 4;"B";AT I+1,X;" ": BEEP .1,0: NEXT I: PRINT
 AT 6,X;" ": LET P=P+(25 AND A(X-7)): LET A(X-7)=0: P
RINT AT 1,8;FN P$(P)
 190 GO TO 210
 200 PRINT AT 16,X; PAPER 1;" ";AT 16,X1; INK 7;"A":
LET X=X1
 210 IF RND*10000>P+7000 THEN  GO TO 150
 220 LET A=INT (RND*16)+1
 230 PRINT AT A(A)+6,A+7;"D";AT A(A)+7,A+7;"C": LET A
(A)=A(A)+1: IF A(A)=9 THEN  GO TO 250
 240 GO TO 150
 250 IF P>R THEN  FLASH 1: LET R=P: PRINT AT 1,27;FN
P$(R): FOR I=30 TO 0 STEP -1: BEEP .1,I: NEXT I: FLAS
H 0
 260 PRINT AT 11,3;"DESEA INTENTARLO OTRA VEZ?"
 270 IF INKEY$="S" THEN  GO TO 70
 280 IF INKEY$="N" THEN  GO TO 10000
 290 GO SUB 270
 300 LET X=INT (RND*253)+1: LET Y=INT (RND*174)+1
 310 PLOT X,Y: DRAW 0,-Y: PLOT X,Y: DRAW 0,175-Y: PLO
T X,Y: DRAW -X,0: PLOT X,Y: DRAW 255-X,0
 320 LET J=175-Y: LET K=255-X: IF K<J THEN  LET J=K
 330 PLOT X,Y: DRAW J,J
 340 LET J=175-Y: LET K=X: IF K<J THEN  LET J=K
 350 PLOT X,Y: DRAW -J,J
 360 LET J=Y: LET K=255-X: IF K<J THEN  LET J=K
 370 PLOT X,Y: DRAW J,-J
 380 LET J=Y: LET K=X: IF K<J THEN  LET J=K
 390 PLOT X,Y: DRAW -J,-J
 400 RETURN
 410 DATA 24,126,102,90,102,126,126,126
 420 DATA 24,56,96,124,30,6,28,24
 430 DATA 8,8,60,60,90,129,129,66
 440 DATA 8,8,8,8,8,8,8,8
 450 DEF FN P$(Y)=CHR$ 19+CHR$ 0+CHR$ 17+CHR$ 1+CHR$
16+CHR$ 7+("000"+STR$ Y)(LEN STR$ Y TO )
```

DATOS EN PROGRAMA

Ya hemos visto en capítulos anteriores dos de las formas posibles de asignar valores a las variables: la primera de ellas, a través de la sentencia **LET**, y la segunda, mediante **INPUT**. Por ambos procedimientos, se reemplaza el valor que hasta el momento tiene una variable, del tipo que sea, para adoptar en adelante el indicado por las sentencias de asignación (**LET** o **INPUT**).

En el primer caso, el nuevo valor asignado a la variable en cuestión es el indicado por la constante o variable situada a la derecha del símbolo de igualdad. En el segundo, la asignación se realiza por el valor que introducimos mediante **INPUT**, en el mismo momento de la ejecución del programa, y es, por supuesto, el valor aceptado por el teclado.

Además de estos dos sistemas que acabamos de repasar, existe un tercero todavía desconocido para nosotros. Su objetivo es recuperar datos contenidos en el propio programa, que han sido situados en el lugar deseado, por medio de la palabra clave **DATA**.

En realidad, la operación de inclusión de datos en el programa se realiza a través de la sentencia **DATA**, mientras que la recuperación se efectúa por medio de la sentencia **READ**; pero está claro que la utilización de la segunda, implica haber hecho antes uso de la primera, por lo cual ambas instrucciones quedan estrechamente vinculadas, dado que ninguna de ellas tiene sentido sin la otra. Nos encontramos ante un nuevo caso de «simbiosis» informática, similar a la descrita en el tratamiento de bucles mediante **FOR** y **NEXT**.

Si tratamos de encontrar un sentido práctico a este conjunto de sentencias, nos damos cuenta en seguida de su utilidad para incluir datos fijos en el programa, como pueden ser nombres de datos a pedir por medio de un **INPUT**, la lista de los meses del año, etc... Así pues, dada su considerable importancia, pasemos al estudio exhaustivo de este sistema de lectura de datos.

LA SENTENCIA DATA

La palabra clave **DATA** puede incluirse en cualquier parte del programa, siendo su misión infor-

En la asignación de valores, entran en juego tres sistemas: LET, INPUT *y* READ-DATA-RESTORE.

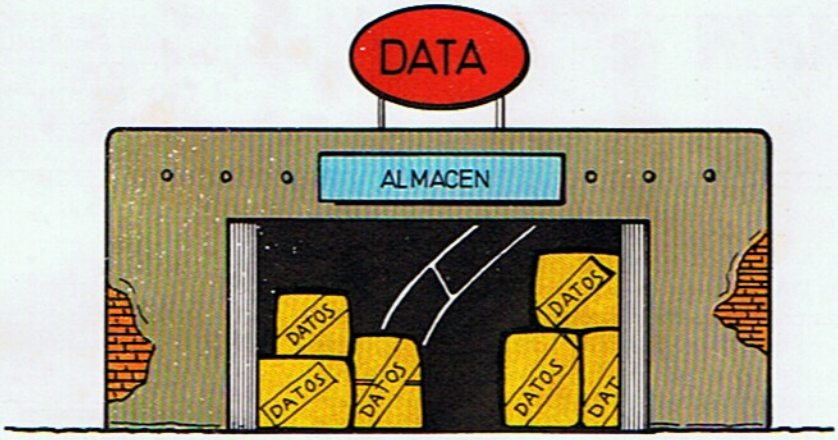
Una línea que comienza con la sentencia **DATA**, provoca una interpretación especial, calificando la línea en cuestión como datos, que serán accesibles mediante alguna sentencia **READ** situada en el programa.

*

RESTORE altera el orden secuencial de ejecución de las lecturas de los elementos de las **DATA**, de forma similar a como actúa una sentencia **GO TO** en las líneas de instrucciones convencionales.

*

Cuando tratamos de leer un elemento superior al último de los contenidos en las **DATA**, nos encontramos con el mensaje de error **E Out of DATA**.



Las sentencias DATA *se emplean para el almacenamiento de datos en el programa.*

En una sentencia **DATA** pueden combinarse los dos tipos de datos, numéricos y de cadena, separados por comas (,).

La lectura de los elementos de las **DATA** se produce de forma secuencial, a partir del número de línea más bajo hasta el último, elemento por elemento.

mar al intérprete del BASIC que la línea que la contiene no debe ser interpretada como una línea de programa convencional. Su significado en Castellano es DATOS, y precisamente eso es lo que pretende comunicar al intérprete BASIC: «No te preocupes por esta línea, pues sólo contiene datos que serán leídos mediante **READ**».

De algún modo, existe un cierto paralelismo de concepción entre esta sentencia, y otra que ya nos es muy familiar: **REM.** También en el caso de **REM,** las líneas que comienzan con esta palabra clave no son interpretadas como líneas convencionales por el ordenador, aunque existe una diferencia fundamental en cuanto a operatividad entre una sentencia y otra.

REM hace que el ordenador ignore por completo el contenido de la línea, ya que su función es meramente de ayuda a la documentación del programa. **DATA,** sin embargo, provoca una interpretación diferente, calificando la línea en cuestión como datos, que serán accesibles mediante alguna sentencia **READ** situada en el programa.

A pesar de que, como ya hemos dicho, las líneas de **DATA** pueden situarse en cualquier parte del programa, existen dos criterios preferentes de colocación, por motivos de claridad y modularidad del mismo.

El primer criterio es el de emplazar todas las líneas de **DATA** agrupadas al final del programa. El segundo, consiste en situarlas inmediatamente a continuación de la línea que contiene su sentencia **READ** correspondiente, es decir, emparejadas con la instrucción que hace uso directo de ellas.

La elección de uno u otro sistema queda al libre albedrío del programador, aunque no debemos olvidar que se trata de simples recomendaciones referentes a la forma y no al fondo, y que el programa funcionará exactamente igual si hacemos caso omiso de ellas.

Ahora bien, sin duda nos preguntaremos, que si la situación de **DATA** puede ser tan anárquica como ha podido parecer, ¿cómo sabe el ordenador qué dato leer al ejecutar un **READ?** ¿acaso puede haber un solo dato en el programa? Evidentemente, la segunda pregunta cae por su propio peso; ¡no sería de mucha utilidad poder almacenar un solo dato! Aclaremos pues la primera cuestión.

Si bien no importa el punto del programa en que se sitúen las **DATA,** sí es fundamental el orden en que se encuentren. Veremos esto más claramente con un ejemplo: supongamos que tenemos un programa de sólo dos líneas...

```
10 INPUT "NUMERO";N
20 PRINT N*N
```

Este programa funcionará a la perfección, solicitándonos la introducción de un número, y escribiendo a continuación su cuadrado; ahora bien, ¿qué sucede si cambiamos los números de las instrucciones?

```
23 INPUT "NUMERO";N
57 PRINT N*N
```

Absolutamente nada; todo funcionará igual de bien. Sin embargo, alterar el orden de las instrucciones, aun conservando su numeración inicial, puede conllevar efectos catastróficos...

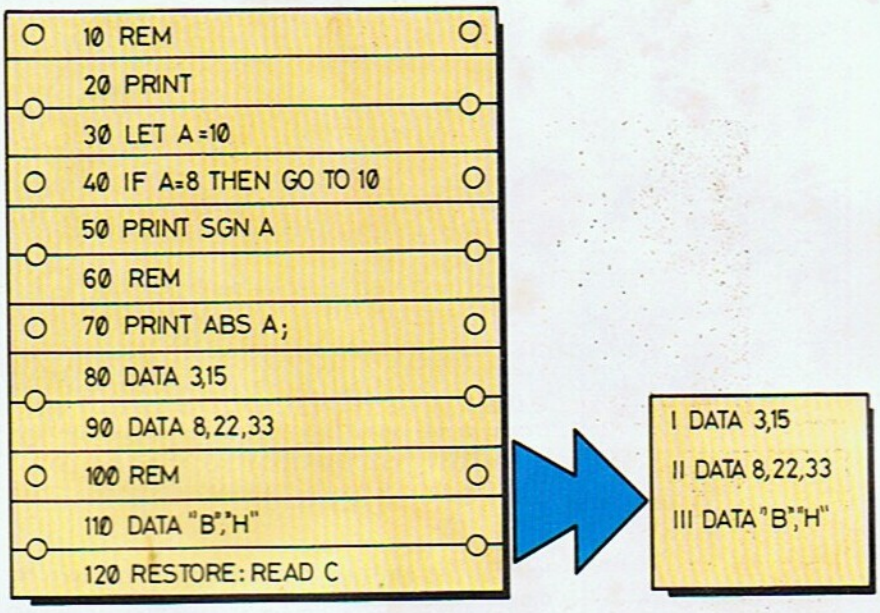
```
10 PRINT N*N
20 INPUT "NUMERO";N
```

Entre READ, DATA *y* RESTORE *se produce un caso de «simbiosis» informática, similar al bucle* FOR-NEXT.

De esta forma tan gráfica, habremos podido comprobar que lo importante en el programa no es el número de las líneas de instrucción, sino el orden en que estas aparezcan. Esto mismo sucede con las **DATA.**

Cuando por primera vez se ejecuta una sentencia **READ** (lectura de **DATA**), el intérprete BASIC comienza a buscar desde el principio del programa, hasta encontrar la primera sentencia **DATA**; entonces, lee el primero de los datos, y «recuerda» en qué posición se encuentra éste, mediante un elemento que denominaremos PUNTERO DE DATOS *(DATA POINTER).* Al realizar una siguiente lectura **(READ)**, el intérprete buscará el próximo dato a partir del último localizado (puntero de datos), de forma que ya no vuelva a leer otra vez el primero de los datos, almacenando la nueva posición en el puntero. Esta operación se repetirá siempre al ejecutar un **READ.**

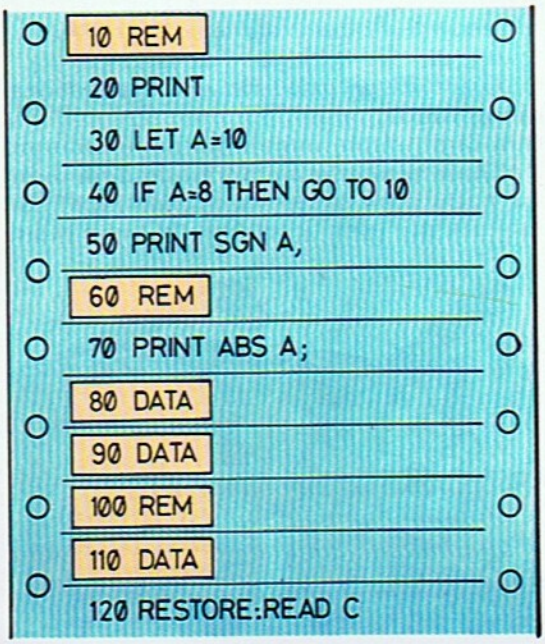
Ahora que ya conocemos el funcionamiento general del sistema de **DATA**s, entraremos de lleno en la sintaxis y características especiales de estas sentencias.



El lugar del programa en que se sitúen las sentencias DATA *es indiferente, lo que importa es el orden de colocación.*

USO COMBINADO DE READ Y DATA

La primera puntualización acerca de la sentencia **DATA**, es que dentro de una misma línea pueden incluirse varios datos, separándolos con comas (,). La segunda, es que los datos no numéricos deben incluirse entrecomillados. Lo veremos más claramente en el siguiente programa de ejemplo:




```
10 REM - MESES Y DIAS
20 PRINT "M E S E S","DIAS"
30 FOR I=1 TO 12:READ X$:PRINT AT
I+1,0;X$:NEXT I
40 FOR I=1 TO 12:READ X:PRINT AT
I+1,16;X:NEXT I
50 DATA "ENERO","FEBRERO","MAR
ZO","ABRIL"
60 DATA "MAYO","JUNIO","JULIO","A
GOSTO"
70 DATA "SEPTIEMBRE","OCTU
BRE","NOVIEMBRE","DICIEMBRE"
80 DATA 31,28,31,30,31,30,31,31,30,31,
30,31
```

Como breve comentario a este programa, podemos decir que, en la línea 20, se imprime la cabecera de las dos columnas de datos, en la línea 30 se sitúa el bucle de lectura e impresión de los nombres de los meses del año y, por último, en

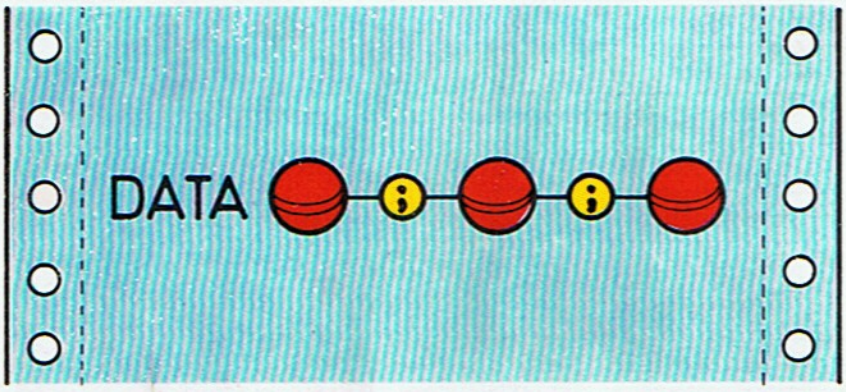
Las sentencias DATA, *al igual que* REM, *son ignoradas por el intérprete BASIC al pasar por ellas.*

¡!

RESTORE puede ir seguida de un argumento numérico, que indica el número de línea donde se quiere recomenzar la lectura de datos mediante **READ**; de carecer de este argumento, se produce una restauración total al primer elemento de la primera línea de **DATA**.

*

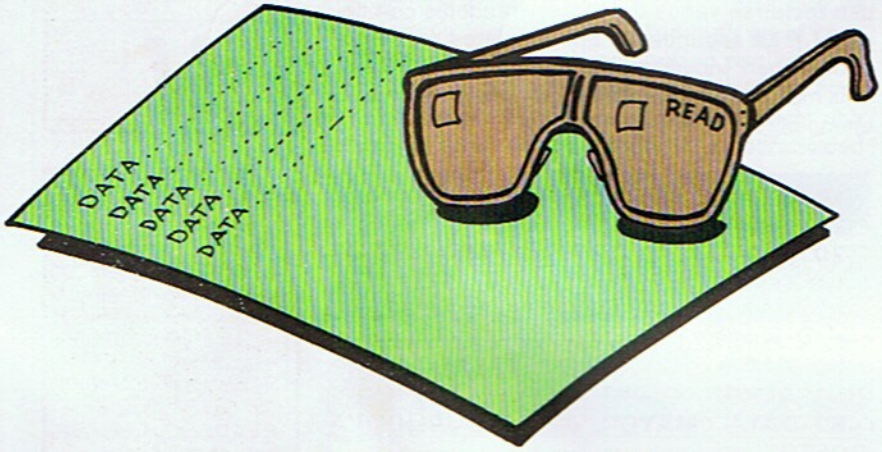
En la lectura de **DATA**, hemos de tener cuidado para que el tipo de dato leído corresponda con el de la variable a asignar.



Dentro de una misma línea DATA, *se pueden incluir varios elementos separados por comas* (,).

la línea 40, se hace lo propio con el número de días de cada mes.
Hemos dejado deliberadamente en el olvido las líneas 50 a 80, que contienen los datos (**DATA**) para las sentencias **READ**.

La sentencia READ *se utiliza para la lectura de* DATAs.



Las sentencias DATA *pueden cobijar tanto datos numéricos, como de cadena, o combinaciones de ambos.*

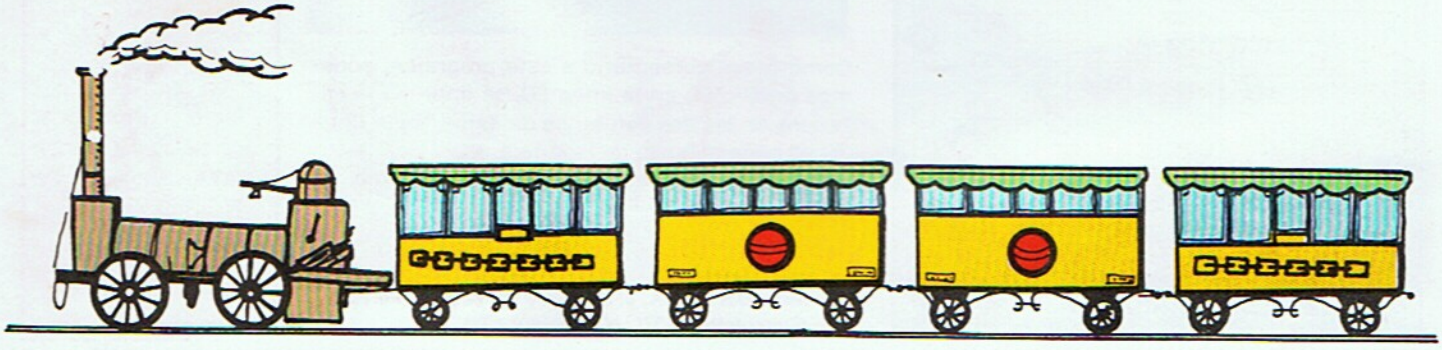
En ellas debemos significar dos cosas: la primera, la calificación del tipo de dato a leer por medio de las comillas, y la segunda, el hecho de que el intérprete, en su discurrir secuencial por las líneas de programa, ha pasado por encima de las **DATA** sin producirse ningún tipo de inconveniente.
Este hecho se debe a que, como hemos dicho anteriormente, son calificadas como líneas sólo utilizables por las sentencias **READ** del programa, en este caso, las de las líneas 30 y 40. Como comprobación, podemos incluir una sentencia **STOP** en la línea 45, obteniendo idénticos resultados:

```
10 REM - MESES Y DIAS
20 PRINT "M E S E S","DIAS"
30 FOR I=1 TO 12:READ X$:PRINT AT
I+1,0;X$:NEXT I
40 FOR I=1 TO 12:READ X:PRINT AT
I+1,16;X:NEXT I
45 STOP
50 DATA "ENERO","FEBRERO","MAR
ZO","ABRIL"
60 DATA "MAYO","JUNIO","JULIO","A
GOSTO"
70 DATA "SEPTIEMBRE","OCTU
BRE","NOVIEMBRE","DICIEMBRE"
80 DATA 31,28,31,30,31,30,31,31,30,31,
30,31
```

Como también habremos notado, el tipo de variable que se utiliza para **READ**, debe corresponder con el tipo de dato leído de la **DATA**, puesto que de no ser así, el programa se detendría con un error **C Nonsense in BASIC**. Así pues, recapitulando, el proceso completo que se desencadena durante la ejecución de un **READ** es:

1. Lectura del próximo elemento de **DATA**.
2. Actualización del puntero de datos y
3. Asignación del dato leído a la variable indicada tras **READ**.

Otra característica interesante de la sentencia **DATA**, es la de poder mezclar los dos tipos de datos, numéricos y de cadena, en una misma línea. Esto nos va a permitir codificar de nuevo el pro-



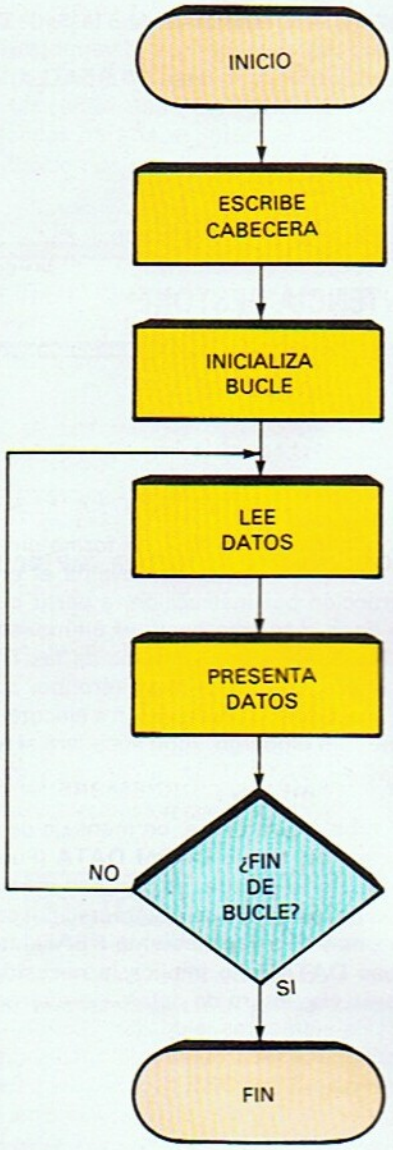


Diagrama de flujo del programa "MESES Y DIAS".

Varios READ, *al igual que* DATA, *se pueden encadenar mediante comas* (,).

grama, de una forma más simple, para escribir los datos en horizontal, y no en vertical, como en el programa del ejemplo anterior:

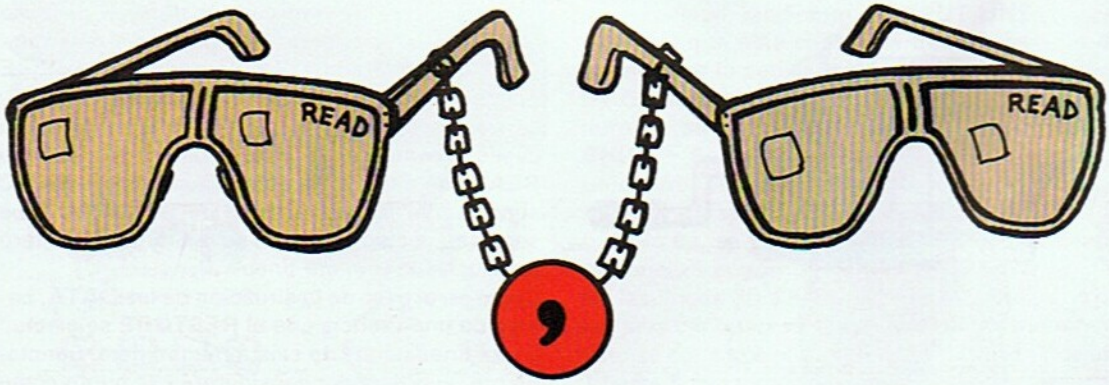
```
10 REM - MESES Y DIAS
20 PRINT "M E S E S","DIAS""
30 FOR I=1 TO 12:READ X$,X:PRINT X$,X:NEXT I
40 DATA "ENERO",31,"FEBRERO",28,"MARZO",31
50 DATA "ABRIL",30,"MAYO",31,"JUNIO",30
60 DATA "JULIO",31,"AGOSTO",31,"SEPTIEMBRE",30
70 DATA "OCTUBRE",31,"NOVIEMBRE",30,"DICIEMBRE",31
```

El nuevo programa contiene además una modificación sustancial, que ha eliminado la antigua línea 40: en la línea 30 podemos apreciar una sentencia **READ** con sus elementos separados por comas. Del mismo modo que la sentencia **INPUT** permite la petición de más de un dato, separando éstos por una coma (,), puede efectuarse la lectura de varios elementos por medio de **READ**, separando los elementos también mediante coma (,).

Antes de continuar, conviene hacer una aclaración acerca del comportamiento de **READ** con **DATA**s de más de un elemento. Realmente este es el caso más frecuente, y **READ** los trata como si estuvieran en líneas **DATA** separadas, es decir, el puntero de datos no señala la última línea leída mediante **READ**, sino el último dato leído. Debido a esto, la estructura siguiente: **10 DATA 31,28,30,31**, es totalmente equivalente a...

```
10 DATA 31
20 DATA 28
30 DATA 30
40 DATA 31
```

De igual modo, el tratamiento de **READ** con separación de comas, es el mismo que si se encon-



¡!

Pueden incluirse varios elementos dentro de una misma línea de **DATA**, separándolos con comas (,).

*

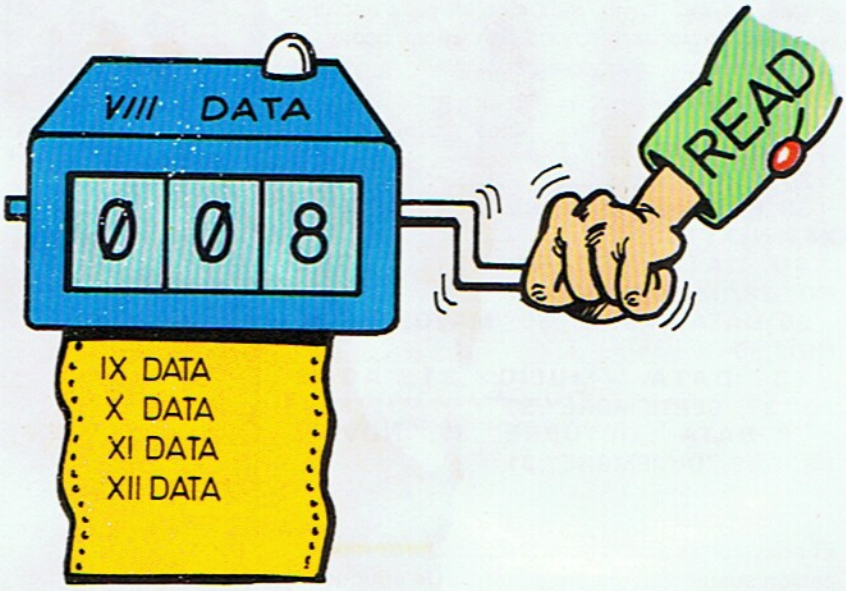
Por medio del conjunto de instrucciones **READ** y **DATA** pueden recuperarse datos contenidos en el propio programa. Ambas sentencias quedan estrechamente vinculadas, puesto que ninguna de ellas tiene sentido sin la otra.

*

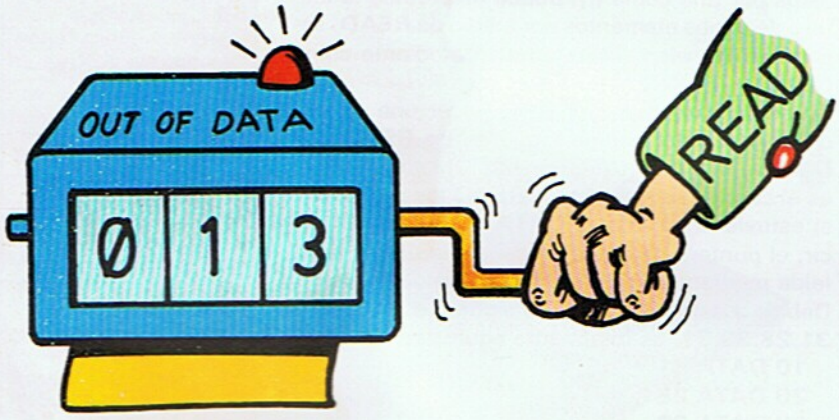
La palabra clave **DATA** puede incluirse en cualquier parte del programa, siendo su misión la de informar al intérprete del BASIC, que la línea que la contiene no debe ser interpretada como una línea de programa convencional.

*

Varias sentencias **READ** pueden ser encadenadas mediante comas (,).

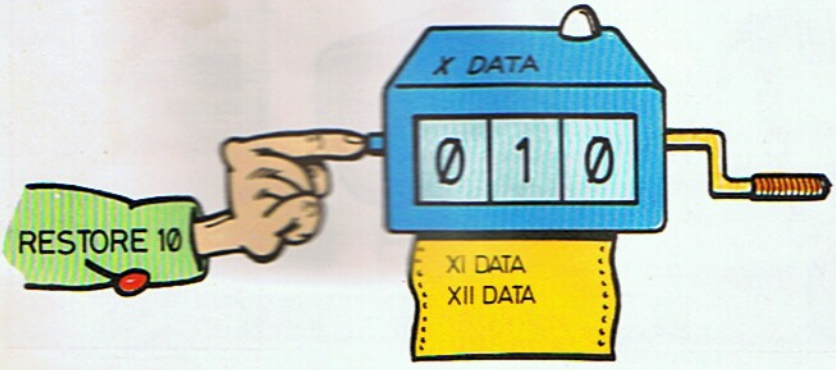


Cada vez que se ejecuta un READ, *se actualiza el puntero de datos.*



Al intentar efectuar más READ*s que* DATA*s tiene el programa, produciremos un error del tipo* E Out of DATA.

El RESTORE *puede operar a partir del número de línea que deseemos.*



traran en líneas diferentes: el dato leído de **DATA** es asignado exclusivamente a la variable que corresponda. Por tanto, la línea **10 READ X,Y,** tiene un efecto idéntico a...

```
10 READ X
20 READ Y
```

LA SENTENCIA RESTORE

Según lo dicho hasta ahora, de forma similar a como el intérprete del BASIC evalúa el programa, instrucción por instrucción, a partir del número de línea más bajo hasta el último, se produce la lectura de los elementos de las **DATA.** Cuando el programa llega a su término, por carecer de una siguiente instrucción a ejecutar, este se detiene. Sin embargo, ¿qué sucederá si agotamos todas las **DATA** de un programa y ensayamos un nuevo **READ?**

Obtendremos, lógicamente, un mensaje de error, en este caso del tipo **E Out of DATA** (Fuera de **DATA**). Lo cierto es que, en algunas ocasiones, y debido a necesidades de programación, se precisa leer una y otra vez mediante **READ** las mismas líneas **DATA;** ello implica la necesidad de volver atrás el puntero de datos, algo parecido a «desleer» los elementos leídos por **READ.**

Para solucionar este problema, el BASIC dispone de la sentencia **RESTORE** (RESTAURA). Esta altera el orden secuencial de ejecución de lecturas, dentro de los elementos de **DATA,** de forma similar a la alteración que en el discurrir de un programa, produce una sentencia **GO TO.** Para ser más exactos, tiene la propiedad de situar el puntero de datos al comienzo del programa, de igual modo que si acabara de ser ejecutado mediante **RUN.**

RESTORE va aún más lejos, y nos permite colocar el puntero de datos en el número de línea que deseemos, si bien no a partir del dato concreto que queramos. La sentencia **RESTORE** puede ir seguida o no de un argumento numérico, correspondiente al número de línea donde se desea recomenzar la lectura de datos mediante **READ;** de carecer de este argumento, el BASIC supone que la restauración de las **DATA** debe ser total, debido a lo cual, se le otorga al puntero el valor de la primera línea.

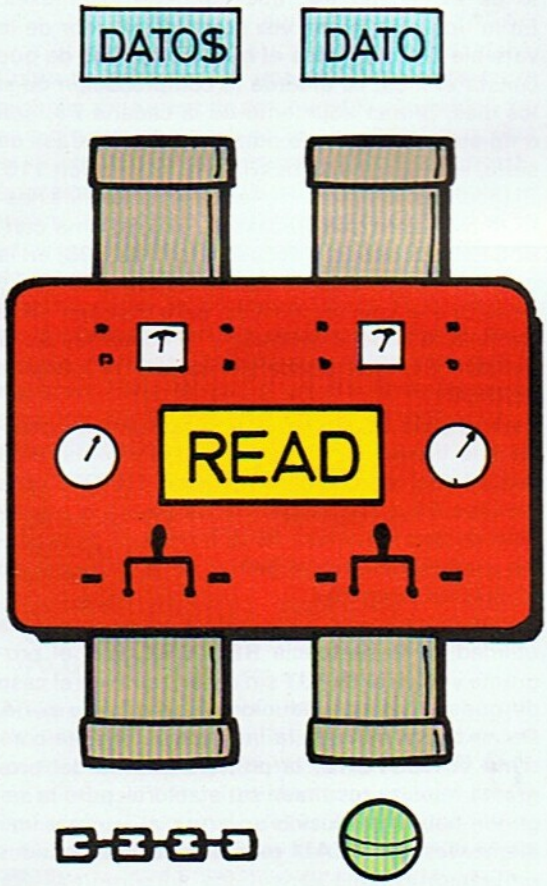
Como en el caso de la situación de las **DATA,** carece de importancia que el **RESTORE** se ejecute a una línea donde no exista ningún dato. Simplemente, el ordenador considerará ese punto como

inicio de la búsqueda de una **DATA** a leer **(READ).**

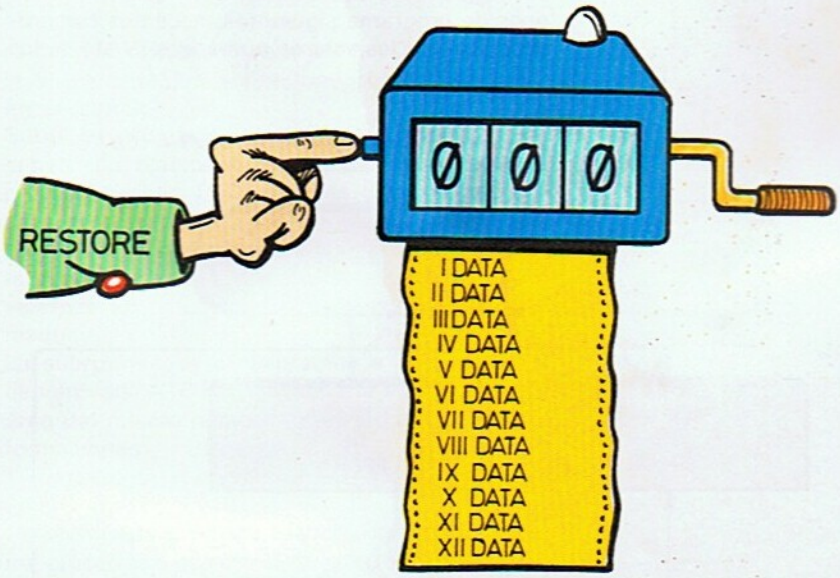
Veremos ahora un ejemplo de utilización conjunta de las sentencias **READ, DATA,** y **RESTORE,** empleadas para la depuración de errores, en la introducción de una fecha por el teclado.

```
  10 REM DEPURACION DE FECHAS - J.M.LOPE
Z MARTINEZ
  20 INPUT 'Fecha:'; LINE F$: IF LEN F$<
>6 THEN  GO TO 110
  30 FOR I=1 TO 6
  40 IF F$(I)<'0' OR F$(I)>'9' THEN  GO
TO 110
  50 NEXT I
  60 IF NOT VAL F$( TO 2) OR NOT VAL F$(
3 TO 4) OR VAL F$(3 TO 4)>12 THEN  GO TO
 110
  70 RESTORE
  80 FOR I=1 TO VAL F$(3 TO 4): READ D:
NEXT I
  90 IF VAL F$( TO 2)>D THEN  GO TO 110
 100 LET E$='': GO TO 120
 110 LET E$='ERROR!'
 120 PRINT F$,E$: GO TO 20
 130 DATA 31,29,31,30,31,30,31,31,30,31,
30,31
```

Al efectuar **READ,** *hay que vigilar que el tipo de variable a asignar coincida con el tipo de dato a leer.*



Este pequeño programa resuelve, aunque no de forma exhaustiva, el problema de depurar la entrada de una fecha que deseamos sea coherente, imprimiendo por cada nueva entrada dos columnas, en la primera de las cuales figura el dato introducido mediante la sentencia **INPUT,** y en la segunda un indicativo de si éste es o no correcto. Decimos que la depuración no es exhaustiva, por-



La sentencia **RESTORE,** *se emplea para restaurar a su posición inicial el puntero de datos.*

que no tiene en cuenta para nada el año, es decir, admite desde el 00 hasta el 99, no reparando por tanto en si éste es bisiesto o no; consecuentemente, siempre se admite la fecha del 29 de febrero. Para que podamos comprobar el funcionamiento del programa, deberemos introducir la fecha en el formato día (2 dígitos), mes (2 dígitos) y año (2 dígitos). Así por ejemplo, la fecha 15 de julio de 1908, se expresaría como: 150708.

Dicho esto, entraremos en el comentario del programa. En la línea 20, se acepta la fecha **F$** desde el teclado, mediante un **INPUT LINE.** Recordaremos que esta sentencia tiene la ventaja de no efectuar la petición del dato alfanumérico entre comillas, y que para interrumpir el programa durante su ejecución, debemos teclear **CAPS SHIFT + 6.** En la propia línea 20, se establece un control para averiguar si la longitud de la cadena de caracteres entrada es o no diferente de 6. Si lo es, se salta a la instrucción 110, donde se canaliza la gestión de fechas con error.

En las líneas 30 a 50 se comprueba si alguno de los seis caracteres entrados no fuera numérico. Esto se consigue con un bucle **FOR-NEXT** desde 1 hasta 6, comprobando carácter a carácter si és-

¡!

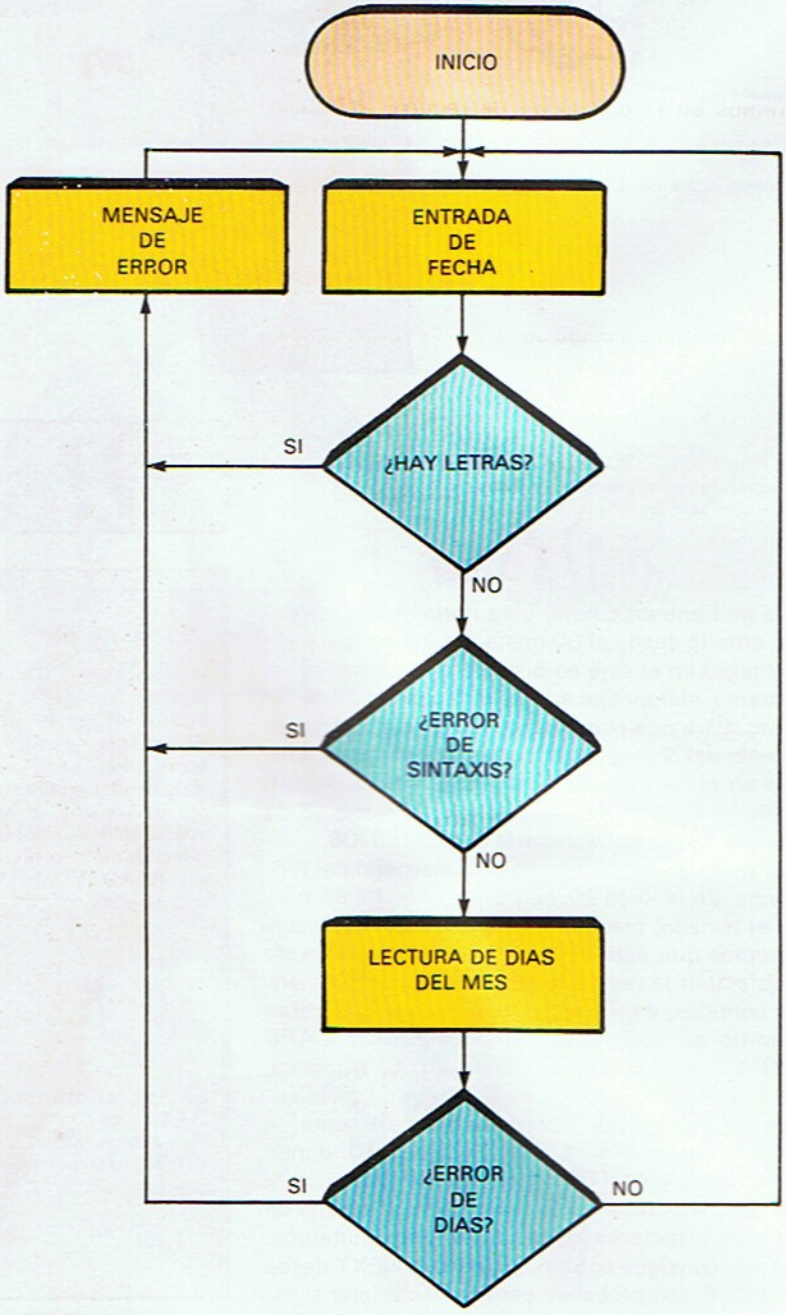
Las líneas de **DATA** pueden situarse en cualquier parte del programa, aunque suele seguirse el criterio de incluirlas al final del mismo, o inmediatamente a continuación de la sentencia **READ** que hace uso de ellas.

*

En las sentencias **DATA** los datos no numéricos deben incluirse entrecomillados.

tos son inferiores a 0 o superiores a 9. Caso de serlo cualquiera de ellos, se salta a la instrucción 110; de no ser así, el programa continúa en secuencia con las siguientes instrucciones.
Ya nos es conocida la potencia de interpretación de la sentencia **VAL**, y es debido a esto que nos vemos obligados a la inclusión en el programa de las líneas 30 a 50. El motivo es que, en las líneas de programa siguientes, hacemos comprobaciones con los valores numéricos (**VAL**) de los tres segmentos de dos caracteres, de los cuales se compone la cadena **F$**; donde los dos primeros caracteres de la cadena indican el día, el tercero y el cuarto el mes, y los dos últimos (quinto y sexto), el año.
Si hubiéramos permitido que se filtrara algún carácter no numérico, por ejemplo **A4**, en las posiciones tercera y cuarta de la cadena, correspondientes al mes, al calcular el **VAL** de este segmento de la cadena, hubiéramos obtenido un error **2 Variable not found** (variable no encontrada), puesto que el intérprete BASIC supone que, al no ser numérico el segmento analizado, debe buscar el valor de la variable **A4** en la memoria del ordenador, lo cual motiva la aparición del error.
En la línea 60 se establece la comprobación de si el día y el mes son cero y, en el caso del mes, si además no es superior a 12. Al detectarse cualquiera de estos errores, puesto que los operadores relacionales son de tipo **OR**, se bifurca la ejecución a la instrucción 110.
En la línea 70 se ejecuta una sentencia **RESTORE**. Esta, al carecer de número de instrucción como argumento, inicializa el puntero de lectura de **DATA** a la primera línea. Por otra parte, en la línea 80, hemos codificado un bucle **FOR-NEXT**, que efectúa tantas lecturas **READ** de la **DATA**, como indica el **VAL** del segundo segmento de la cadena **F$**, que contiene los meses. En la línea 90, y una vez obtenido el valor de la variable **D**, que indica el número de días de que consta el mes, se efectúa la comprobación de si los días, primer segmento de la cadena **F$**, son o no superiores a este número máximo. Caso de serlo, el programa se desvía a la instrucción 110.
Si todas las comprobaciones de error hechas hasta el momento han fracasado, el programa continúa en secuencia y accede a la línea 100, en la cual se anula el contenido de la variable **E$**, y se salta a la instrucción 120, para imprimir en la pantalla la cadena tecleada y el contenido de la variable **E$**, que puede ser nada o el mensaje "ERROR!".
Para el caso de los saltos que se puedan producir a la línea 10, por detección de errores en la depuración, se asigna el valor de "ERROR!" a la variable **E$** y, seguidamente, se pasa a la impresión en dos columnas de la línea 120, para volver inmediatamente al **INPUT** de la 20, cerrando el ciclo de programa.
Debemos observar, para valorar en su medida la utilidad de la sentencia **RESTORE**, que el programa vuelve al **INPUT** sin cesar, tanto en el caso de que se hayan producido errores como si no. De no haber incluido la instrucción 70 que contiene el **RESTORE**, la primera pasada del programa hubiera resultado satisfactoria, pero la segunda habría producido un error, al intentar leer elementos por **READ** más allá de los incluidos en la instrucción 130.

Diagrama de flujo del programa "DEPURACION DE FECHAS".



GRAFICOS ANIMADOS

EL último capítulo nos sirvió de introducción al dinamismo a pantalla completa, y tuvimos la oportunidad de comprobar la eficacia de una de las técnicas a este fin: el dinamismo por intermitencia. Sin embargo, quedó pendiente el estudio de otras dos técnicas, a cual más importante, que concentraban su acción independientemente sobre el área de atributos, y sobre el archivo de imagen. Pasaremos a continuación a dar cumplida información de ambas.

CONTROL DE ATRIBUTOS

Como ya sabemos, la imagen compuesta en la pantalla del televisor por nuestro ordenador, no es a efectos de programación un todo único; ésta se divide en dos componentes absolutamente diferentes: la forma de la pantalla y el color de la misma. La técnica de dinamismo por atributos, se basa precisamente en la animación de uno solo de estos componentes: el color.

Efectivamente, la alteración rápida de los colores de la pantalla, aun sin afectar a la forma de la misma, es decir, los gráficos, textos, etc., que la integran, tiene de por sí un efecto dinámico de gran interés. Vamos a comprobarlo mediante un sencillísimo ejemplo:

```
10 INK 0
20 PRINT AT 11,12;"MENSAJE"
30 PAUSE 5
40 INK 7
50 PRINT AT 11,12;"MENSAJE"
60 PAUSE 5
70 GO TO 10
```

Aparentemente, el contenido escrito en el archivo de imagen se altera, apareciendo y desapareciendo, sin embargo, tanto en la línea 30 como en la 50 escribimos un idéntico mensaje. Efectivamente, el contenido del archivo de imagen no ha variado; no obstante, hemos alterado el del área de atributos, cambiando el color de escritura del mensaje, alternando la tinta 0 (negro), con la 7 (blanco, que sobre fondo blanco se hace imperceptible).

Sería por tanto interesante poder someter a toda la pantalla, o al menos a gran parte de ella, a un proceso similar. Dado que las dimensiones del área de atributos son de relativa consideración (768 bytes), y que precisamos un efecto prácticamente simultáneo sobre todo el área a tratar, nos veremos en la obligación de recurrir al código máquina.

La subrutina que presentamos a continuación, denominada ATRIBUTOS, somete a cada byte del área del mismo nombre, a las siguientes transformaciones:

* Incrementa el código de la tinta, cuidando de no sobrepasar el valor 7.
* Incrementa el código de fondo, teniendo la misma precaución que en el paso anterior.
* Deja inalteradas las cualidades de **BRIGHT** (brillo) y **FLASH** (intermitencia).

Todo esto puede que no nos parezca de un excesivo interés, sin embargo, merece la pena que introduzcamos el siguiente programa de demostración; su efecto sin duda nos sorprenderá.

```
 10 REM ATRIBUTOS - C.DE LA OSSA & F.LOPEZ MARTINEZ
 20 DATA 1,0,88,17,0,3,10,60,230,7
 30 DATA 103,10,198,0,230,56,111,10,230,192
 40 DATA 133,132,2,3,27,122,179,32,233,201
 50 CLEAR 32499: FOR I=0 TO 29
 60 READ J
 70 POKE 32500+I,J
 80 NEXT I
 90 BORDER 1: PAPER 1: CLS
100 FOR I=0 TO 7
110 PRINT  PAPER I;"
"
120 NEXT I
130 PRINT AT 2,14; PAPER 2; INK 9;"RUN";AT 3,5; PAPE
R 3;"ENCICLOPEDIA PRACTICA";AT 4,9; PAPER 4;"DEL  SPE
CTRUM"
140 FOR I=0 TO 19
150 READ J,K: LET L=I-8*INT (I/8)
160 PRINT AT J,K; PAPER L;"   ";AT J+1,K; PAPER L;"
 ";AT J+2,K;"  "
170 NEXT I
180 DATA 9,15,10,13,11,11,12,9,13,7,14,5
190 DATA 15,7,16,9,17,11,18,13,19,15
200 DATA 18,17,17,19,16,21,15,23,14,25
210 DATA 13,23,12,21,11,19,10,17
220 FOR I=8 TO 10: CIRCLE  INK 7;27,11,I: NEXT I
230 FOR I=8 TO 10: CIRCLE  INK 7;228,11,I: NEXT I
240 FOR I=8 TO 10: CIRCLE  INK 7;27,88,I: NEXT I
250 FOR I=8 TO 10: CIRCLE  INK 7;228,88,I: NEXT I
260 PRINT AT 15,12;"ATRIBUTOS"
270 LET I=5
280 RANDOMIZE USR 32500
290 LET I=I+1: IF I=8 THEN  LET I=0
300 BORDER I
310 PAUSE 5
320 GO TO 280
```

¡!

Al efectuar **POKE**s al área de atributos, debemos de tener mucho cuidado para no exceder su límite inferior, puesto que a continuación se encuentra el programa en BASIC, y su deterioro puede ser irremediable.

BITS

Una forma muy rápida de traducir una palabra a su forma bajo-alto es efectuar **RANDOMIZE <número a codificar>**. Acto seguido, el **PEEK** de la dirección 23670 contendrá su peso bajo y **PEEK** de 23671, el alto. He aquí un programa de utilidad basado en esta facultad.

```
10 REM BAJO-ALTO
20 INPUT "NUMERO ? ";N
30 IF N<0 OR N>65535 THEN GO TO 20
40 RANDOMIZE N
50 PRINT "BAJO: ";PEEK 23670
60 PRINT "ALTO: ";PEEK 23671
```

*

Las posiciones investigadas, corresponden a la variable del sistema **SEED**, cuya misión es servir de «semilla» en la extracción de números aleatorios.

*

Para proteger el código máquina de la sobreescritura por el BASIC, utilizamos la sentencia **CLEAR** seguida de un argumento numérico, con el cual se señala la última dirección utilizable por el BASIC. Este sistema presenta el gran inconveniente de borrar también las variables. Para proteger el código máquina, sin necesidad de borrar las variables, podemos acceder directamente a la variable del sistema **RAM TOP**: **RANDOMIZE <valor de tope de memoria>: POKE 23730,PEEK 23670: POKE 23731,PEEK 23671.**

Ahora que ya hemos salido de nuestro asombro, y somos conscientes de la gran utilidad de la subrutina de atributos, podemos pasar a un estudio más detallado de la misma. Comencemos por el listado de ensamblador de la página siguiente. Esencialmente, se utilizan dos pares de registros: el BC y el DE. El primero de ellos sirve de puntero del byte a tratar en el área de atributos, y por tanto comienza siendo 22528 (primera posición de dicha zona), que va incrementándose según se va completando el tratamiento de los bytes. El par DE se utiliza como contador del número de bytes a analizar, y comienza siendo 768 (total de bytes del área de atributos), valor que se va decrementando conforme se van alterando los bytes; cuando DE llega a cero, la subrutina ha terminado su trabajo.

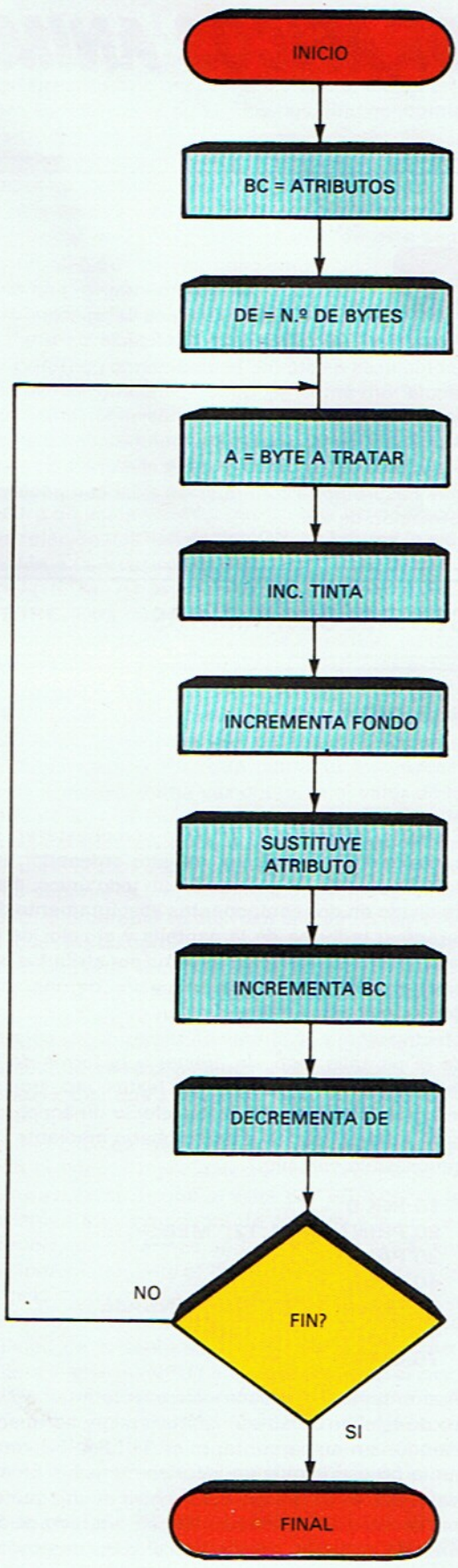
Quizás pensemos que la utilización de DE podía haber sido evitada, simplemente utilizando también BC para investigar si se ha llegado al final de la pantalla (23295); sin embargo, utilizar DE concederá una mayor flexibilidad a la rutina. Veamos por qué.

Si deseamos tratar todos los bytes de la pantalla, la subrutina no precisará ningún cambio, sin embargo, con su actual programación, es enormemente sencillo alterar el número de bytes a tratar (sólo con hacer los **POKE**s correspondientes en bajo-alto, al quinto y sexto byte de la rutina). Del mismo modo, cambiar el primer byte a analizar, también será muy fácil, puesto que sólo tendremos que depositar la primera dirección a tratar (en bajo-alto), en el segundo y tercer byte de la rutina. Si llevamos a cabo estas modificaciones, hemos de tener cuidado para que el byte inicial más el número de bytes alterados no sobrepase el último byte de la zona de atributos (23295).

Por otra parte, todos los que tengamos unos mínimos conocimientos de código máquina, sabremos que incrementar o decrementar pares de registros es igualmente sencillo, sin embargo, a la hora de las comprobaciones, es mucho más fácil investigar si un par de registros es cero (un simple OR entre los registros que lo componen), que si ha alcanzado determinado valor.

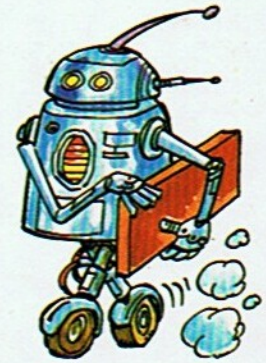
Para la adopción de la rutina por nuestra propia cuenta, podemos utilizar el siguiente programa:

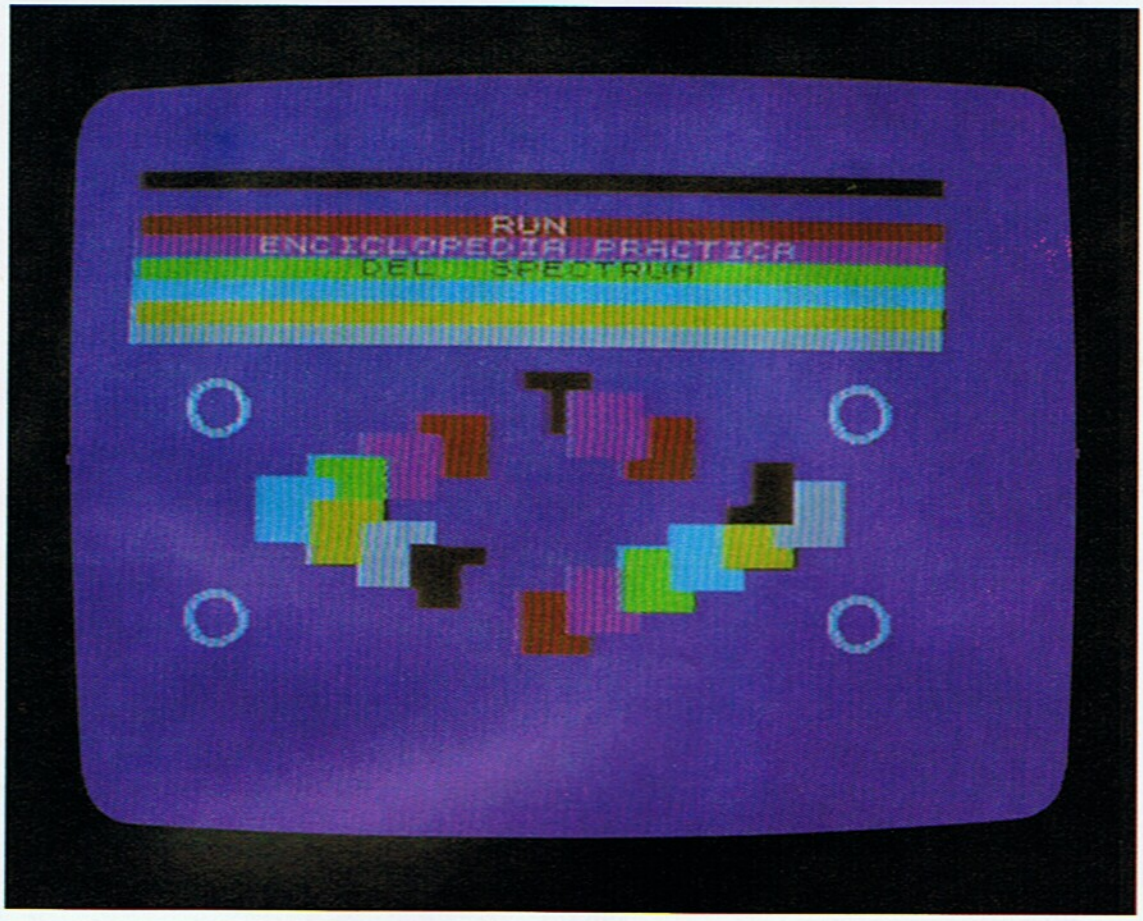
```
10 REM RUTINA DE ATRIBUTOS - C. DE LA OSSA & F. LOPEZ MARTINEZ
20 CLEAR 32499
30 DATA 1
40 DATA 0,88
50 DATA 17
60 DATA 0,3
70 DATA 10,60,230,7,103,10,198,8,230,56,111,10,230,192,133,132,2,3,27,122,179,32,233,201
```



SUBRUTINA DE ATRIBUTOS

ETIQUETA	OBJETO	FUENTE	COMENTARIO
ATTR	1 0 88	LD BC,22528	;Carga HL con la primera dirección a ;tratar.
	17 0 3	LD DE,768	;Carga DE con el número de bytes a tratar.
LOOP	10	LD A,(BC)	;Inicia el ciclo de tratamiento car gando el ;acumulador con el byte a tratar.
	60	INC A	;Incrementa de uno A, y por tanto, el có-;digo de tinta.
	230 7	AND A,7	;Pone a cero los bits del 3 al 7, y deja inal-;terados los restantes (del 0 al 2); Así ;pues, si la tinta pasó a ocho, queda a cero ;y si no, con el incremento correspon-;diente.
	103	LD H,A	;Guarda el estado actual del acumulador ;(nuevo código de tinta) en H.
	10	LD A,(BC)	;Carga el acumulador con el byte a anali-;zar. Por tanto, el valor anterior se pierde, ;por lo cual fue guardado previamente ;en H.
	198 8	ADD A,8	;Suma 8 al acumulador, es decir, incre-;menta de uno el código de fondo.
	230 56	AND 56	;Pone a cero los bits 0, 1, 2, 6 y 7, y deja ;inalterados los restantes (del 3 al 5). Así ;pues, si el código de fondo pasó a 8 que-;da a cero, si no, mantieel incremento.
	111	LD L,A	;Guarda el acumulador en L, es de cir, el ;código de fondo.
	10	LD A,(BC)	;Carga el acumulador con el byte a ana-;lizar; de ahí que el valor anterior fuera ;preservado en L.
	230 192	AND 192	;Pone a cero los bits del 0 al 5, y deja inal-;terados los restantes (6 y 7). Gracias a ;esto, anula los bits de tinta y fondo, tra-;tados anteriormente, y deja en el acumu-;lador sólo los de brillo e intermitencia, ;que no serán modificados.
	133	ADD A,L	;Suma L al acumulador, es decir, añade a ;brillo e intermitencia ante riores, el nue-;vo código de fondo creado por la su-;brutina.
	132	ADD A,H	;Suma H al acumulador, por tanto, añade ;al A anterior al código de tinta.
	2	LD (BC),A	;Deposita el nuevo atributo en la posición ;correspondiente.
	3	INC BC	;Incrementa de uno la posición a analizar.
	27	DEC DE	;Decrementa de uno el número de bytes ;que quedan por tratar.
	122	LD A,D	;Carga el acumulador con D, preparándo-;lo para la comprobación de fin de tarea.
	179	OR E	;Suma lógica de D y E. Su resultado será ;comprobado en la siguiente instrucción.
JUMP	32 233	JR NZ LOOP	;Si D OR E distinto de cero, DE es distinto ;de cero, por lo cual volve mos al comien-;zo del ciclo de tra tamiento.
END	201	RET	;Si D OR E es cero implica que DE=0, por ;tanto, no quedan bytes por tra tar, y po-;demos devolver el control al BASIC.





¿?

Con la expresión QUEDAR COLGADO, se designa usualmente la pérdida de control sobre el ordenador.

*

En la lectura de listados, el símbolo #, suele denominarse: almohada, *number* (pronunciado *namber)* o *hush* (pronunciado *hash).*

*

En la lectura de listados, los elementos seguidos de un argumento entre paréntesis, como las variables suscritas o las partes de cadenas, se suelen leer como: <elemento> DE <argumento entre paréntesis>. Así por ejemplo, **G (H,L)**, se lee, "G DE H COMA L".

```
80 FOR I=0 TO 29
90 READ A
100 POKE 32500+I,A
110 NEXT I
```

Según el lugar donde deseemos situar la subrutina, deberemos sustituir el **POKE 32500+I,A** de la línea 100, y por supuesto, el **CLEAR** de protección de la rutina, establecido en la línea 20, que siempre debe estar al menos una posición antes que la de comienzo de la rutina. Una vez ejecutado este programa, el código máquina quedará almacenado en la memoria, y podrá ser utilizado en cualquier momento mediante **RANDOMIZE USR <dirección de carga>** (en el ejemplo, **RANDOMIZE USR 32500).**

Para finalizar la adaptación de la rutina a nuestra propia conveniencia, los bytes de la **DATA** en la línea 50, corresponden, en peso bajo-peso alto, al primer byte a tratar en el área de atributos, inicialmente 22528 (0+88*256=22528). Por otra parte, la **DATA** de la línea 60 contiene el número de bytes a analizar, expresador por el mismo sistema (bajo-alto), que inicialmente especifican 768 (3*256=768).

Así por ejemplo, si deseamos que sólo estén afectados por la rutina de atributos los últimos dos tercios de la pantalla, deberemos sustituir la línea 20 del programa de demostración por **20 DATA 1,0,89,17,0,2,10,60,230,7.** Puesto que el primer byte a tratar (primero del segundo tercio) es 22784 (22784=0+ 256*89), y el área son 16 líneas de 32 caracteres, es decir, 512 bytes (512=0+256*2).

Por último, y para comprender completamente el trabajo realizado por la subrutina de atributos, podemos diseñar una subrutina que realice la misma función en lenguaje BASIC; la lentitud será muy grande, pero eso ayudará a que podamos apreciar perfectamente todos los cambios que se operan. Para utilizar esta subrutina propuesta, sólo hemos de añadirla al programa de demostración, y sustituir la llamada al código máquina de la línea 280, **(RANDOMIZE USR 32500),** por el acceso a la subrutina: **280 GOSUB 1000.**

```
1000 LET BC=22528
1010 LET DE=64
1020 LET A=PEEK BC
1030 LET A=A+1
1040 LET M=7: GO SUB 1200
1050 LET H=A
1060 LET A=PEEK BC
1070 LET A=A+8
1080 LET M=56: GO SUB 1200
```

```
1090 LET L=A
1100 LET A=PEEK BC
1110 LET M=192: GO SUB 1200
1120 LET A=A+L
1130 LET A=A+H
1140 POKE BC,A
1150 LET BC=BC+1
1160 LET DE=DE-1
1170 IF DE<>0 THEN  GO TO 1020
1180 RETURN
1190 REM OPERACION LOGICA A AND M
1200 LET B=A: GO SUB 1310: LET A$=B$
1210 LET B=M: GO SUB 1310: LET M$=B$
1220 FOR J=1 TO 8
1230 LET A$(J)=STR$ (VAL A$(J)*VAL M$(J)
)
1240 NEXT J
1250 LET A=0
1260 FOR J=1 TO 8
1270 LET A=A+VAL A$(J)*2^(8-J)
1280 NEXT J
1290 RETURN
1300 REM DECIMAL => BINARIO
1310 LET B$=''
1320 IF B<2 THEN  GO TO 1360
1330 LET C=INT (B/2): LET R=B-C*2
1340 LET B$=STR$ R+B$
1350 LET B=C: GO TO 1320
1360 LET B$='0000000'+STR$ B+B$
1370 LET B$=B$(LEN B$-7 TO )
1380 RETURN
```

PESO BAJO - PESO ALTO

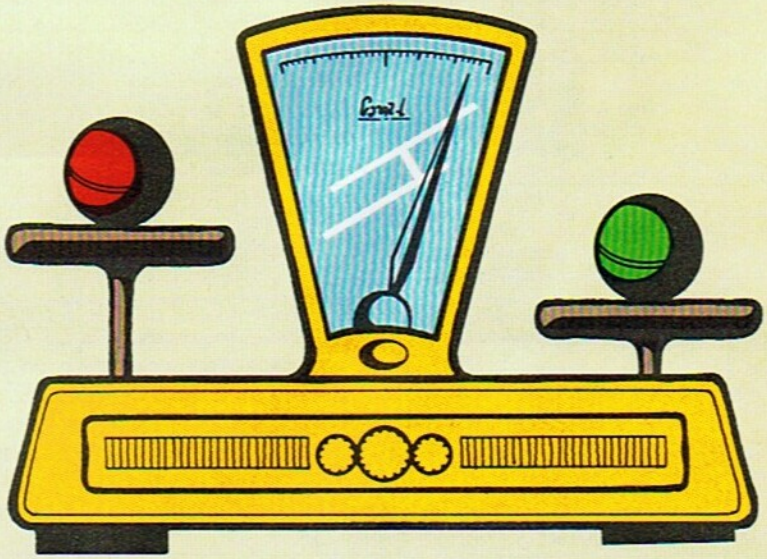
Como ya sabemos, la unidad de memoria más ampliamente difundida es el BYTE, compuesto por ocho bits, y en base al cual se mide la capacidad de memoria de un ordenador. Puesto que un bit puede portar dos informaciones distintas (0 ó 1), un byte puede codificar hasta 256 datos diferentes (desde 0 hasta 255). No obstante, en algunas ocasiones, esta cantidad de información es insuficiente, por lo cual tenemos que utilizar alguna unidad superior de información.

Esta es, la PALABRA (en inglés *WORD),* que se entiende compuesta por dos bytes, y por ende, por 16 bits. Así pues, la palabra puede portar 65536 informaciones diferentes. Es por ello, por ejemplo, que nuestro ordenador puede llegar a tener en el total de su memoria, precisamente 65536 bytes (64 K); y esto no ha sido por capricho, sino porque esta es la máxima cantidad de información representable en una palabra (dos bytes). Ahora bien, la forma de leer un byte de la memoria, o depositarlo en ella, es bien sencilla, gracias a que disponemos de sentencias BASIC especializadas en ello: **PEEK** y **POKE**. Sin embargo, ¿cómo hacer para trabajar con palabras? Primero es necesario que tengamos claro un concepto acerca del «peso» de los dígitos en una cifra. Si tomamos el número decimal 23 y le damos la vuelta, conseguiremos un valor completamente diferente (32), pese a que hemos empleado los mismos dos dígitos. Esto se debe a que el valor de un número, no sólo depende de sus dígitos, sino también de la posición de éstos en la cifra; cuanto más a la izquierda se encuentre el dígito, más influye en la magnitud del número. Se dice entonces, que los dígitos de más a la izquierda son más SIGNIFICATIVOS, o también, que tienen mayor PESO.

Con los dos bytes que configuran una palabra, sucede exactamente lo mismo: no es igual la palabra 34 - 201, que la 201 - 34. Es importante que se nos indique por tanto, cual de los dos bytes es el más significativo. Lo más habitual es que las palabras se expresen en forma de PESO BAJO PESO ALTO; por tanto, siempre debemos suponer que se encuentran en este formato, salvo que se haga una especificación al contrario.

Una vez sabido cuál es el peso bajo y cual el alto, la decodificación de la información que contiene la palabra es bien simple: el peso bajo, mas el producto del peso alto por 256. Así por ejemplo, la palabra 34 - 201 (bajo-alto), equivale a 34+256*201=51490.

La lectura de una palabra es justo la operación inversa; el peso alto se obtendrá como la parte entera del número a codificar dividido entre 256, y el peso bajo, como dicho número menos el producto de 256 por el peso alto. Siguiendo el mismo ejemplo, el peso alto de 51490 es la parte entera (cociente) de 51490/256=201, y el peso bajo, 51490-256*201=34. Esta operación realizada en BASIC sería: **ALTO=INT (NUMERO/256): BAJO=NUMERO-256*ALTO.**



ACORAZADO

A bordo de un poderoso acorazado, la tensión se hace casi insoportable al contemplar la tripulación como dos mortíferas estelas se aproximan peligrosamente. Con una hábil maniobra, el buque consigue evitar el alcance y lanza cuatro cargas de profundidad encaminadas a la destrucción de su escurridizo enemigo.

Esta, que bien pudiera ser una narración ambientada en la Segunda Guerra Mundial, es la situación en que nos encontramos con el siguiente programa, como capitanes de un poderoso navío: el *H. M. S. RUN* de la *Royal Navy*.

Nuestro objetivo es el hundimiento del mayor número posible de submarinos enemigos. Para ello nos desplazaremos a izquierda y derecha por medio de las teclas 1 y 2, respectivamente, y lanzaremos las mortíferas cargas de profundidad pulsando la tecla 0.

Hay que tener muy en cuenta, que disponemos de un máximo de cuatro cargas cada vez, es decir, cuando se hayan lanzado cuatro cargas tendremos que esperar hasta que alguna de ellas explote, bien por alcanzar el submarino enemigo, o bien por reventar contra el fondo marino, antes de poder arrojar una nueva; las cargas serán lanzadas por la parte izquierda del buque, y podremos saber las disponibles en cada momento fijándonos en el centro de la fila superior de la pantalla.

Contamos con tres oportunidades antes que el Almirantazgo nos retire el mando a causa de nuestra falta de pericia; a modo de recordatorio, en la parte superior derecha de la pantalla, aparece una reproducción de nuestro buque por cada oportunidad que nos queda.

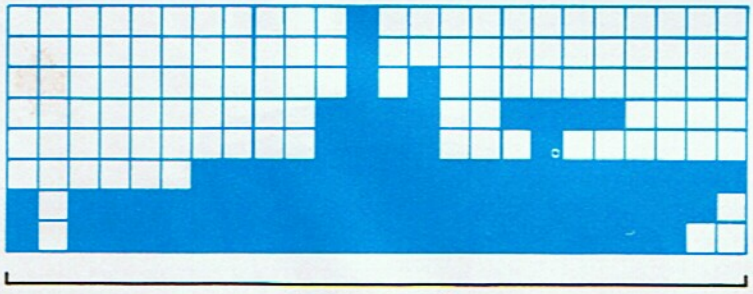
El submarino circula siempre de izquierda a derecha, y no se limita a dar paseos hasta que tengamos a bien sumergirle definitivamente, sino que intenta defenderse por diversos medios:

— surgiendo en cada ciclo a alturas diferentes y
— lanzando torpedos.

Si alguno de estos llegara a alcanzarnos... ¡adiós oportunidad!, y si agotamos todos nuestros intentos, el programa finalizará pidiendo nuestra opinión sobre comenzar una nueva partida. Por otra parte, en la zona inferior de la pantalla (subsuelo marino), se señala permanentemente nuestra puntuación actual y la máxima puntuación obtenida en la jornada.

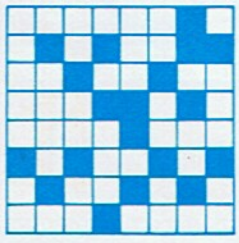
El juego utiliza la subrutina de caracteres gigantes de PSION, y por tanto, una vez grabado el programa BASIC definitivo, deberá grabarse a continuación una copia de dicha subrutina de código

Las dimensiones del submarino son menores que las del acorazado; sólo tres gráficos: E, F y G.

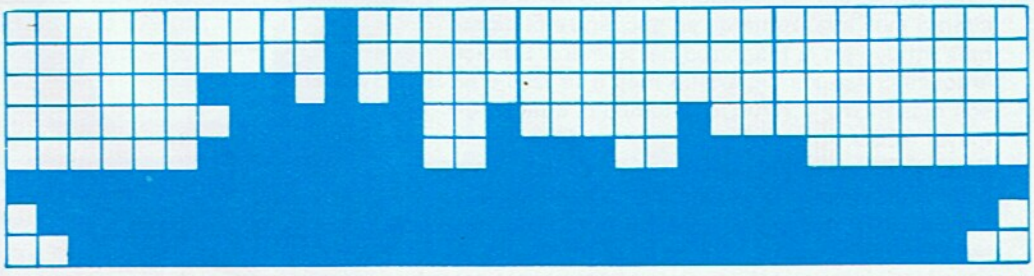


SUBMARINO

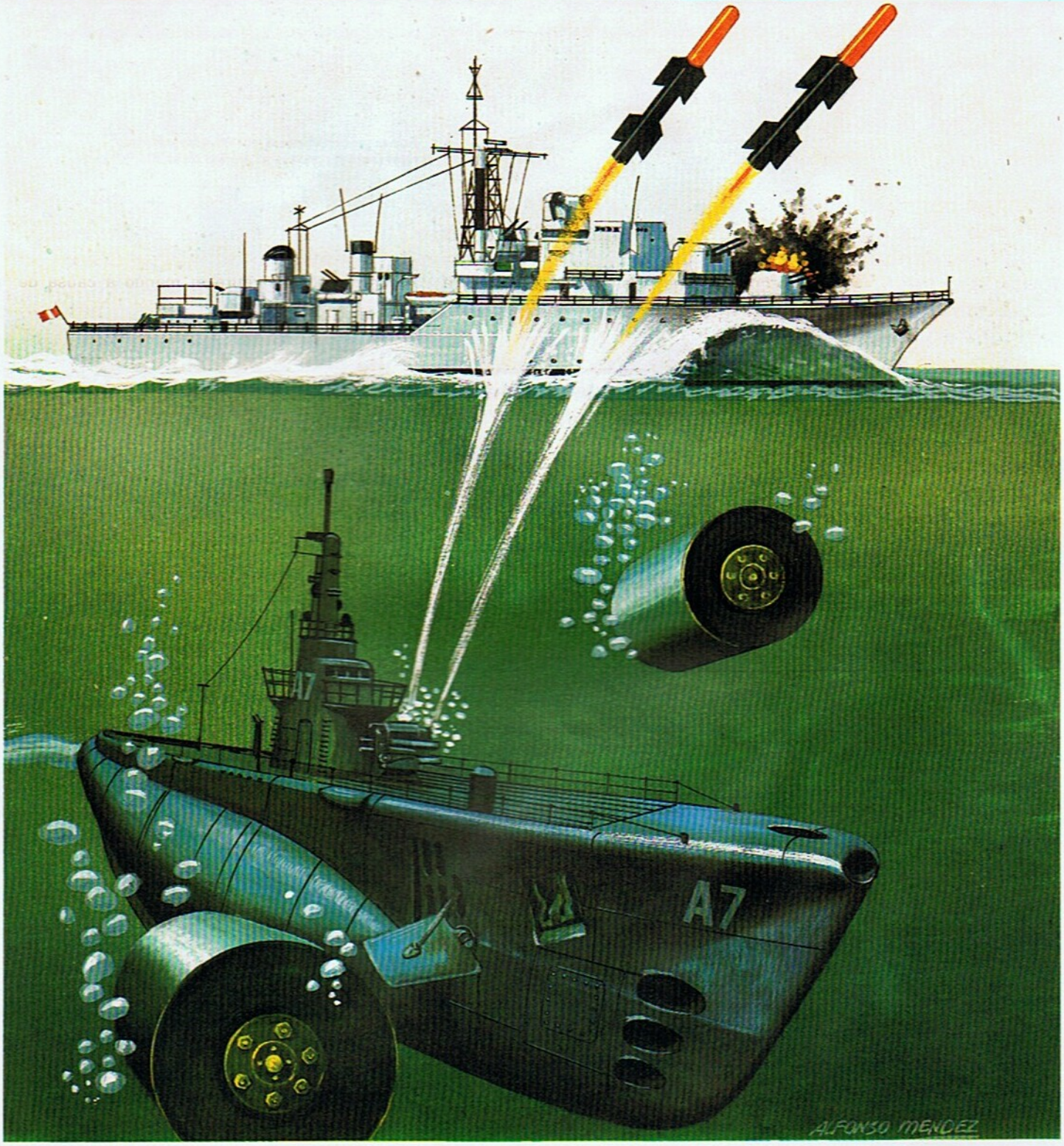
Cuatro son los gráficos que componen la silueta del acorazado: A, B, C y D.



EXPLOSION



ACORAZADO

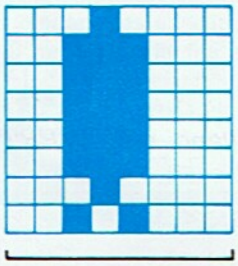

A7
A7
ALFONSO MENDEZ

```
   1 REM * J.M. MAYORAL SERRANO *
  10 LET A$="                                    ": LET B
$="HHHH": LET C$=A$( TO 6)+"ABCD ABCD": LET N=0
  20 BORDER 0: LET A=0: LET B=3: POKE 23658,8
  30 PAPER 1: INK 1: CLS : PAPER 5: PRINT A$;A$;A$
  40 PAPER 4: INK 0: LET Z$="PUNTOS:          RECORD:
"+STR$ N+A$( TO 8-LEN STR$ N): LET XX=0: LET XY=183:
LET XS=1: LET YS=1: GO SUB 650
  50 PAPER 1: INK 4: FOR I=0 TO 2: PLOT 0,1: DRAW 255
,0: NEXT I
  60 RESTORE 70: FOR I=1 TO 28: READ J: READ K: PLOT
J,K: READ J: DRAW J,0: NEXT I
  70 DATA 0,3,40,46,3,25,95,3,12,125,3,12,160,3,42,21
0,3,45
  80 DATA 0,4,38,48,4,20,98,4,8,128,4,6,162,4,39,211,
4,44
  90 DATA 0,5,37,49,5,18,99,5,6,129,5,4,163,5,37,212,
5,43
 100 DATA 0,6,30,52,6,12,101,6,2,130,6,1,167,6,28,217
,6,38
 110 DATA 0,7,20,55,7,6,173,7,16,227,7,20
 120 DIM A(4,2): LET E=1: LET L=0
 130 LET H=13: GO SUB 640
 140 PAPER 4: INK 0: LET Z$=STR$ A: LET XX=64: LET XY
=183: GO SUB 650
 150 LET C=6+INT (RND*15): LET D=-1
 160 PAPER 5: INK 2: PRINT AT 0,16;C$( TO B+5);A$( TO
 4);AT 1,14;B$( TO 4-L);A$( TO 3)
 170 LET D=D+1: IF D=29 THEN  PAPER 1: PRINT AT C,D;A
$( TO 3): GO TO 150
 180 IF NOT ATTR (C,D+2)=11 THEN  GO TO 230
 190 FOR I=1 TO 4
 200 IF A(I,1)=C THEN  GO TO 220
 210 NEXT I
 220 LET D=D-1: GO SUB 600: GO TO 140
 230 PAPER 1: INK 1: PRINT AT C,D;" ": INK 7;"EFG"
 240 IF NOT E THEN  GO TO 270
 250 LET E=INT (RND*C/4): IF E THEN  GO TO 320
 260 LET F=C-1: LET G=D+2
 270 LET F=F-1: PRINT AT F+1,G;" ";AT F+1,G+1;" "
 280 INK 6: PRINT AT F,G;"K";AT F,G+1;"K"
 290 IF F>3 THEN  GO TO 320
 300 PRINT AT F,G;" ";AT F,G+1;" ": LET E=1
 310 IF ATTR (2,G)=42 OR ATTR (2,G+1)=42 THEN  PRINT
AT 2,H+1; PAPER 5; INK 2;"JJJJ": GO SUB 590: PAPER 5:
 PRINT AT 2,H+1;A$( TO 4);AT C,D+1; PAPER 1;A$( TO 3)
: LET B=B-1: GO TO 30
 320 FOR I=1 TO 4
 330 IF NOT A(I,1) THEN  GO TO 390
 340 IF A(I,1)=20 THEN  INK 2: PRINT AT A(I,1),A(I,2)
;"J": GO SUB 590: PRINT AT A(I,1),A(I,2);" ": LET A(I
,1)=0: LET L=L-1: GO TO 390
 350 IF ATTR (A(I,1)+1,A(I,2))=15 THEN  GO SUB 600: G
O TO 140
 360 LET A(I,1)=A(I,1)+1: INK 1: PRINT AT A(I,1)-1,A(
I,2);" "
 370 LET M=0: IF INT (A(I,1)/2)=A(I,1)/2 THEN  LET M=

1
 380 INK 3: PRINT AT A(I,1),A(I,2);("H" AND M)+("I" A
ND NOT M)
 390 NEXT I
 400 LET X$=INKEY$: IF X$<"0" OR X$>"2" THEN  GO TO 5
10
 410 IF X$="1" AND H THEN  LET H=H-1: GO TO 500
 420 IF X$="2" AND H<26 THEN  LET H=H+1: GO TO 500
 430 IF X$<>"0" THEN  GO TO 510
 440 IF L=4 THEN  GO TO 510
 450 LET L=L+1
 460 FOR I=1 TO 4
 470 IF A(I,1) THEN  GO TO 490
 480 LET A(I,1)=3: LET A(I,2)=H: GO TO 510
 490 NEXT I
 500 GO SUB 640
 510 FOR I=1 TO 40-10*L: NEXT I
 520 IF B THEN  GO TO 580
 530 PAPER 1: INK 7: FLASH 1: LET Z$="OTRO INTENTO (S
/N) ?": LET XX=48: LET XY=80: LET YS=2: GO SUB 650: F
LASH 0
 540 LET X$=INKEY$: IF X$<>"S" AND X$<>"N" THEN  GO T
O 540
 550 IF X$="N" THEN  GO TO 10000
 560 IF A>N THEN  LET N=A
 570 GO TO 20
 580 BEEP .002,50: GO TO 160
 590 FOR J=-5 TO -15 STEP -1: BEEP .01,J: NEXT J: RET
URN
 600 PRINT AT A(I,1),A(I,2);" ";
 610 PAPER 1: INK 2: PRINT AT C,D+1;"JJJ": GO SUB 590
 620 PRINT AT C,D+1;A$( TO 3)
 630 LET A(I,1)=0: LET L=L-1: LET A=A+INT (C/(L+1)):
RETURN
 640 PAPER 5: INK 5: PRINT AT 2,H;" ": INK 2;"ABCD";
INK 5;" ": RETURN
 650 LET I=23306: POKE I,XX: POKE I+1,XY: POKE I+2,XS
: POKE I+3,YS: POKE I+4,8: LET I=LEN Z$: FOR J=1 TO I
: POKE 23310+J,CODE Z$(J): NEXT J: POKE 23311+I,255:
LET I=USR 32256: RETURN
 660 REM AUTOEJECUCION
 670 CLEAR 32255
 680 LOAD "LIT"CODE
 690 DATA 0,0,3,1,3,255,127,63
 700 DATA 32,32,168,249,249,255,255,255
 710 DATA 0,0,0,4,231,255,255,255
 720 DATA 0,0,0,0,128,255,254,252
 730 DATA 0,0,0,0,0,3,191,191
 740 DATA 16,16,20,60,60,255,255,255
 750 DATA 0,0,0,240,64,255,254,252
 760 DATA 0,0,0,60,60,0,0,0
 770 DATA 0,0,24,24,24,24,0,0
 780 DATA 2,83,36,26,8,165,74,16
 790 DATA 16,56,56,56,56,56,16,40
 800 RESTORE 690
 810 LET Q$="ABCDEFGHIJK"

 820 FOR I=1 TO LEN Q$
 830 FOR N=0 TO 7
 840 READ A
 850 POKE USR Q$(I)+N,A
 860 NEXT N
 870 NEXT I
 880 RUN
```

máquina, que encontraremos en el programa LA BOMBA.

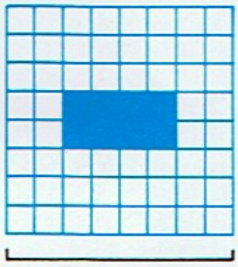
La grabación del programa deberá efectuarse mediante el comando **SAVE "ACORAZADO" LINE 660,** aunque antes debemos recordar sustituir los caracteres que aparecen subrayados en el listado, por los gráficos de las teclas correspondientes, es decir, aquellos que se obtienen pulsando la letra subrayada cuando el cursor se encuentra en modo G **(CAPS SHIFT + 9).**

Los torpedos lanzados por el submarino son representados gracias al gráfico de la K.

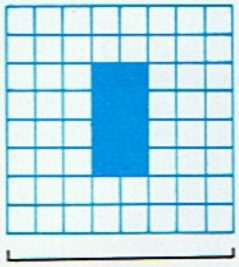


BOMBA

Mediante la utilización alternada de los gráficos H e I, se consigue el desplazamiento y giro de las cargas de profundidad.



CARGA HORIZONTAL



CARGA VERTICAL

VARIABLES SUSCRITAS

RECORDEMOS que hasta ahora hemos hablado de dos tipos de variables, las numéricas y las de cadena o alfanuméricas; siendo el empleo de un tipo de variable u otro discrecional, en función de las características del dato que deba albergar. Las diferencias entre ambos tipos de variables son considerables, y afectan tanto a la forma como al fondo.

El nombre de una variable numérica es una serie de caracteres, y su contenido cualquier número, natural, entero o real, pudiendo efectuarse con ellas cálculos matemáticos, y establecer comparaciones; el nombre de una variable de cadena está formado por un solo carácter seguido del símbolo dólar **($)**, pudiendo ser su contenido cualquier serie de caracteres, numéricos o alfabéticos, y permitiendo operaciones de unión y troceado, además de comparaciones.

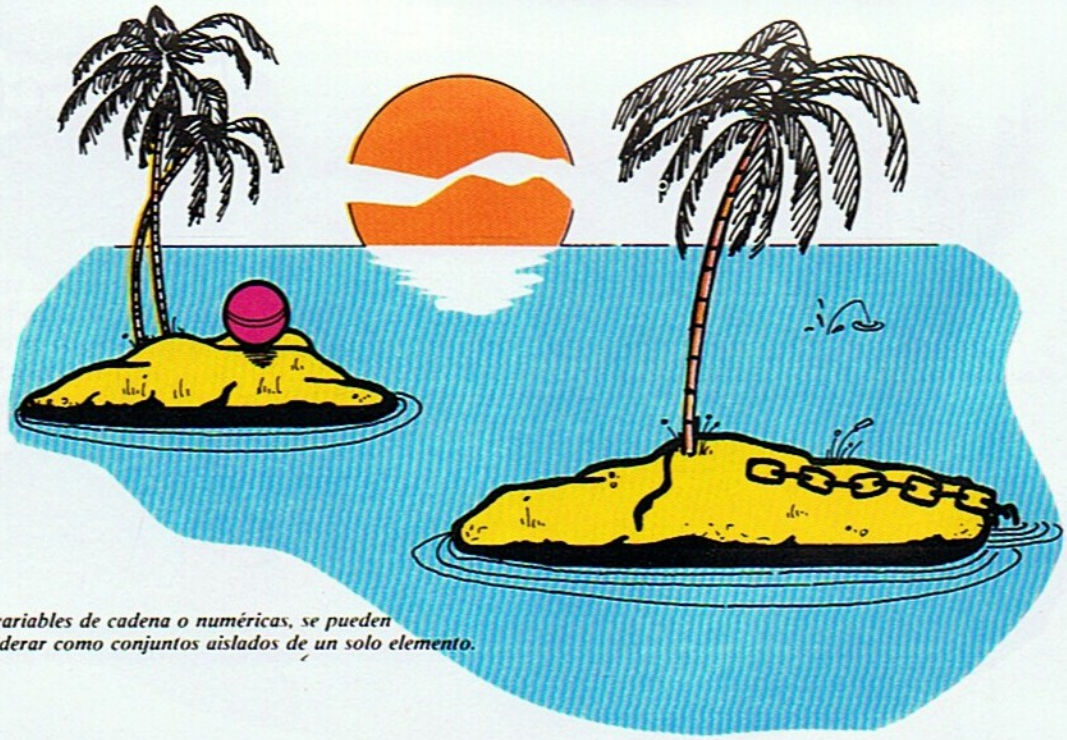
Por la misma forma en que definimos las variables (diferente nombre identificador), cada una de ellas es independiente de las otras, pudiéndolas considerar como conjuntos aislados de un único elemento. Sin embargo, se plantea el problema de definir grupos de variables que nos permitan una doble posibilidad de tratamiento: una individual y otra de «conjunto».

Normalmente, esta situación se produce cuando manejamos un cierto volumen de datos relacionados entre sí. Supongamos, por ejemplo, que queremos averiguar la media aritmética de las notas obtenidas por un alumno, a lo largo de los 10 meses de curso, imprimiendo al final un informe con cada una de las notas y la media del curso.

Para ello, debemos almacenar primero las calificaciones en variables por medio de la sentencia **INPUT**. Podríamos hacerlo pidiendo línea por línea cada una de las calificaciones, y almacenándolas en variables del tipo **N1, N2, ... N10,** pero esta estructura no nos permitiría hacer uso del bucle **FOR NEXT,** y nuestro programa requeriría 10 líneas de **INPUT** y una considerable pérdida de tiempo en la codificación:

```
10 INPUT "NOTA 1";N1
20 INPUT "NOTA 2";N2
   . . .
100 INPUT "NOTA 10";N10
```

Ya hemos visto algunas de las ventajas que se desprenden del uso de los bucles **FOR NEXT.** Sin embargo, nos hemos dejado quizás la más importante: la posibilidad de tratamiento de los elementos de un conjunto.



Las variables de cadena o numéricas, se pueden considerar como conjuntos aislados de un solo elemento.

¡!

Si se intenta asignar a un elemento de una variable suscrita de cadena, un valor de longitud mayor a la preestablecida, éste se trunca, tomándose los *n* primeros caracteres (siendo *n* la longitud máxima), pero no se produce ninguna detención por error.

*

Las matrices de dos dimensiones se suelen conocer como TABLAS.

*

Si se asigna a un elemento de una variable suscrita de cadena, un valor de longitud menor a la preestablecida, éste es completado con espacios en blanco por la derecha, hasta alcanzar dicha longitud.

¡!

Cuando se efectúa un **DIM**, desaparece cualquier variable suscrita de igual nombre que pudiera existir en el momento de la definición, pasando a adquirir la matriz las nuevas características señaladas, como si nunca hubiera existido la variable suscrita anterior.

*

Al definir una variable suscrita numérica, todos sus elementos toman inicialmente el valor cero.

*

Al definir una variable suscrita de cadena, todos sus elementos toman el valor inicial de cadenas de espacios en blanco de la longitud preestablecida.

*

Las variables suscritas, son generalmente conocidas por el nombre de MATRICES.

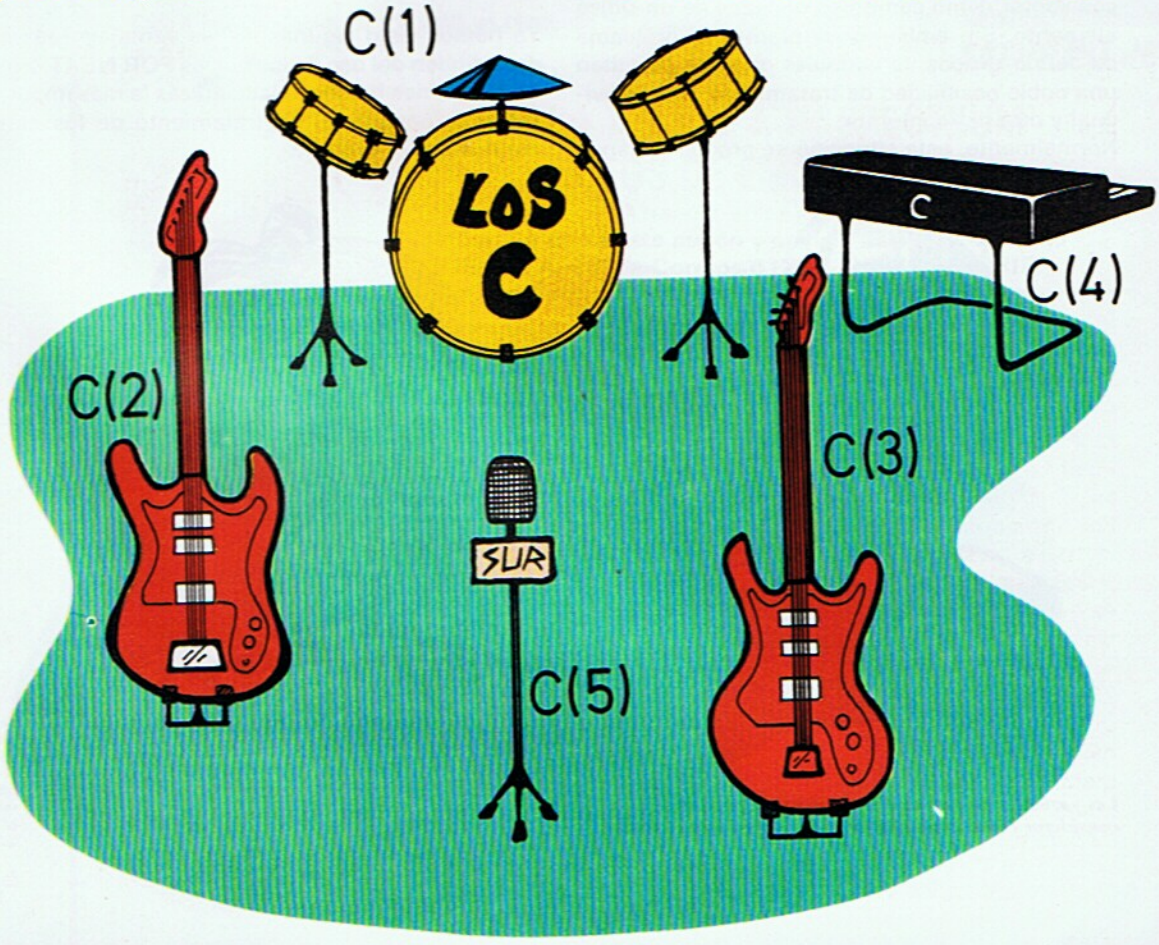
CONJUNTOS DE VARIABLES

No cabe duda que la solución ideal a nuestros problemas es definir un conjunto de variables que se denomina «matriz». Esta matriz va a ser de tipo numérico y su nombre genérico será **N( )**, refiriéndose a cada elemento en particular a través de un «índice», que colocaremos entre paréntesis a continuación del nombre de la matriz.
A modo general, podemos decir que una matriz es un conjunto de elementos de un mismo tipo, a los que nos referimos por medio del nombre del conjunto, seguido de un índice que identifica a cada elemento en particular. Este índice designa a un elemento determinado por su posición relativa frente a los demás, de forma que todos ellos guardan dentro del conjunto un orden determinado que impone el programador.
En el caso concreto de nuestro ejemplo, podríamos llamar N(1) a la nota del primer mes, N(2) a la del segundo, etc... hasta N(10), para el último de los meses.

DEFINICION DE MATRICES

Para que el BASIC pueda manejar una matriz, es necesario que la hayamos definido previamente. Esta definición consiste en determinar, a través de la sentencia BASIC **DIM**, el tipo y número de elementos de que se compone, así como su estructura interna. Se trata pues, de una definición algo más compleja que la que debemos hacer cuando nos referimos por primera vez a un nombre de variable, dentro de un programa, a través de la sentencia **LET**.
La citada operación de definición se conoce como DIMENSIONADO, y produce en el ordenador una reserva del espacio suficiente de memoria para contener la variable; de ahí que en algunas ocasiones, la ejecución de estas sentencias conduzca a un error del tipo **4 Out of memory.**
Respecto a la denominación que pueden adoptar

El BASIC permite tratar determinadas variables como parte de un conjunto.



1	OCTUBRE	
2	NOVIEMBRE	
3	DICIEMBRE	
4	ENERO	
5	FEBRERO	
6	MARZO	
7	ABRIL	
8	MAYO	
9	JUNIO	
10	JULIO	

N( )

En el programa de ejemplo, el conjunto N( ) *agrupa las calificaciones medias obtenidas por un alumno durante los diez meses del curso.*

estos conjuntos de variables, son evidentemente diferentes según tengan un contenido numérico o de cadena. En el primer caso, el nombre estará formado por una sola letra, que siempre irá seguida de los índices de referencia del elemento, encerrados entre paréntesis.

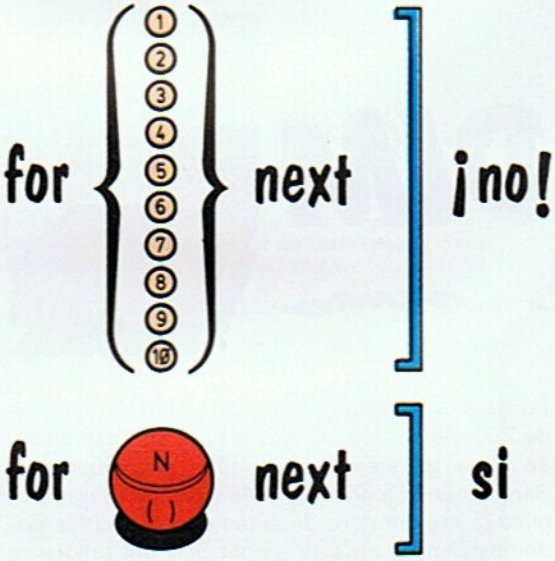
Tratándose de conjuntos de cadena, habremos de añadir a la norma anterior el consabido símbolo dólar (**$**), antes del índice entre paréntesis. En lo referente a este tipo de variables suscritas, existe una restricción muy importante: nunca podemos tener en un mismo momento una variable de cadena y de conjunto de cadenas, que se llamen igual. La razón está bien clara; como ya sabemos, nuestro Spectrum, y en general cualquier artefacto informático, aborrece las ambigüedades, y puesto que el troceado de cadenas se indica encerrando sus parámetros entre paréntesis, podría darse la confusión entre la referencia a un troceado de cadena, o a un índice de variable suscrita. Veamos un ejemplo.

Si suponemos la existencia de una cadena **A$**, y una matriz **A$( )**, puede que queramos conocer el carácter tercero de la cadena **A$**, para lo cual ejecutaríamos **PRINT A$(3)**; sin embargo, idéntica instrucción emplearíamos para conocer el elemento tercero del conjunto **A$( )** (como variable suscrita). Así pues; el ordenador no puede permitir tal confusión en sus electrónicas «neuronas», y se ve obligado a prohibirnos expresamente la utilización de variables de cadena y suscritas que tengan un mismo nombre.

Por otra parte, dado que como denominación de matrices, ya sean numéricas o alfanuméricas, se puede emplear una sola letra, y puesto que, como es habitual en las variables, no existe diferencia alguna entre mayúsculas o minúsculas, el número máximo de variables suscritas de cada tipo es 26 (de la A a la Z); aunque esto no debe resultar un inconveniente insalvable en ningún caso, dado que la posibilidad de manejar múltiples dimensiones con cada matriz, garantiza prácticamente que nunca echaremos en falta variables de este tipo.

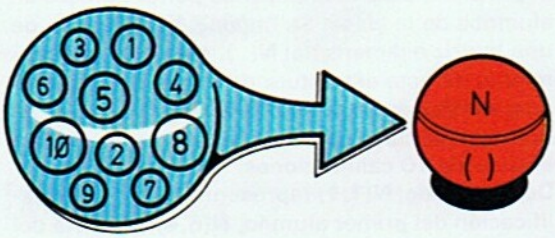
Las matrices alfanuméricas o de cadena, alcanzan usualmente una dimensión. Su definición consiste en indicar, a través de la sentencia BASIC **DIM**, su nombre y, entre paréntesis, su número de elementos y longitud del mayor de ellos, separados por comas. A este tipo de matrices unidimensionales, ya sean numéricas o alfanuméricas, se las denomina también VECTORES o LISTAS, puesto que su desarrollo es lineal. Un vivo ejemplo de matriz unidimensional o vector, es el propuesto anteriormente para la obtención de la nota media en una asignatura. La matriz **N( )** es unidimensional porque, para localizar a un elemento determinado dentro de ella, precisamos de un solo índice, en nuestro caso, la posición relativa de la nota buscada frente a las demás.

Las matrices numéricas admiten más de una dimensión, pero sin embargo no es necesario referenciar la longitud de sus elementos, puesto que



El tratamiento por FOR-NEXT *de variables diferentes, no es posible; sin embargo, sí es válido el de una misma variable suscrita.*

Un conjunto de variables con algún denominador común, pueden reunirse en una sola variable suscrita.



¡!

Las matrices de una sola dimensión se denominan comúnmente LISTAS o VECTORES.

*

Las variables suscritas son conjuntos de variables con un mismo nombre genérico, distinguidas por el ordenador mediante un número entre paréntesis, denominado índice.

*

Las variables suscritas pueden ser numéricas o alfanuméricas.

*

La denominación general de una variable suscrita numérica, debe ser una sola letra; como en el caso de las variables individuales, no existe distinción entre mayúsculas y minúsculas, por lo cual el máximo número de variables suscritas diferentes es 26 (de la A a la Z).

todas las variables numéricas BASIC ocupan lo mismo, con independencia de su contenido. Esto que en un principio nos puede parecer algo difícil de comprender, no nos debe preocupar ahora: es simplemente un hecho, que bastante más adelante estudiaremos.
De esta forma, una matriz numérica queda definida por una sentencia **DIM**, al indicar su nombre y, a continuación y entre paréntesis, su número de elementos y dimensión, separados por una coma. A continuación se muestran algunos ejemplos de definición de diversas matrices:

10 DIM A$(10,20): DIM A(200): DIM B(200,10)

En esta línea de programa se han dimensionado tres matrices. La primera de ellas, **A$**, es de cadena y consta de 10 elementos de longitud máxima 20. La segunda, **A**, es numérica y consta de 200 elementos, en una dimensión, es decir, es

Cada uno de los elementos del conjunto, se distingue de los demás por su índice entre paréntesis.

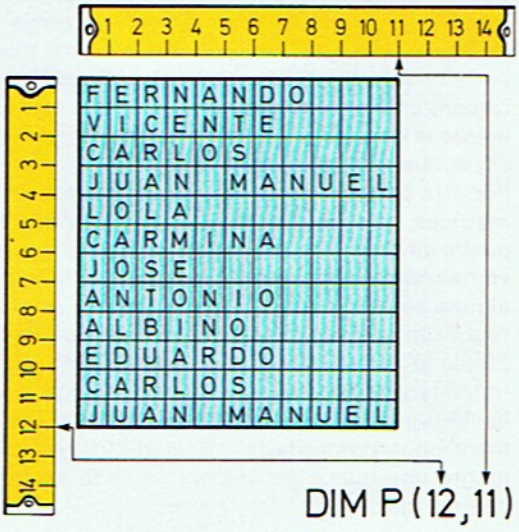
La sentencia DIM *se utiliza para reservar espacio de memoria a una variable suscrita.*

un vector. Por último, **B**, es otra matriz numérica de 200 elementos, cada uno de ellos desdoblado en otros 10, siendo por lo tanto bidimensional.
Para tener un concepto más claro de lo que implica la multidimensionalidad de una matriz, podemos pensar en una ampliación del problema propuesto anteriormente. Supongamos que, en vez de calcular la nota media de las calificaciones de un alumno, deseamos obtener además las de toda la clase.
En este caso, es evidente que podríamos aprovechar gran parte del programa anterior. De hecho, nuestro problema es el mismo que el propuesto al principio, pero multiplicando por el número de alumnos de la clase. Se impone, pues, el uso de una matriz bidimensinal **N( )**, para el almacenamiento de esta estructura de datos, que deberemos definir de la forma **DIM N(50,10)**, suponiendo una clase de 50 alumnos y un período a evaluar de 10 calificaciones.
De este modo, **N(1,1)** representará la primera calificación del primer alumno, **N(6,4)** la cuarta del sexto alumno y, por último, **N(50,10)** la décima del que hace el número cincuenta.
Si ampliamos aún más nuestro ejemplo sobre las notas de los alumnos, haciéndolo extensivo a todas las clases que componen el curso, nos encontramos ante el caso de una matriz tridimensional, que podríamos representar como:

DIM N(8,50,10)

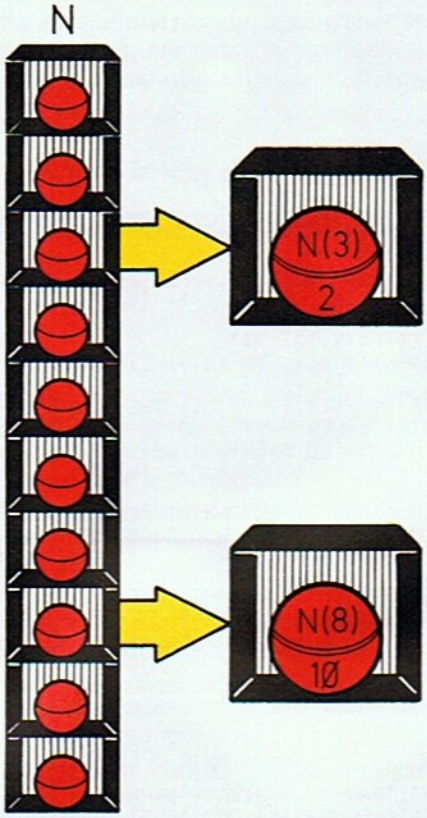
En este caso, sería preciso el empleo de 3 índices para calificar a un elemento determinado, es decir, el número de aula, el número de alumno y de qué calificación se trata.

Las variables suscritas alfanuméricas, se declaran indicando el número de elementos que las componen y su longitud máxima.



CONTENIDO DE LAS MATRICES

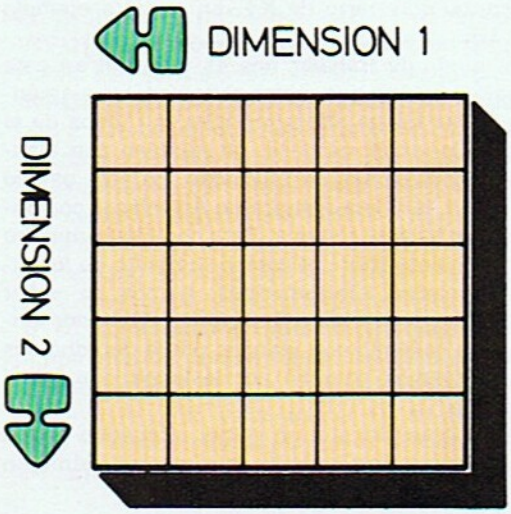
Las matrices, teóricamente, deben definirse una sola vez en el programa. Han de colocarse, por ello, en un lugar concreto del programa por el que sólo se pase una vez. Expliquemos el porqué de ese «teóricamente» del comienzo de párrafo. Cuando una matriz ha sido definida y contiene datos, si se efectúa una nueva definición de la misma matriz a través de otra sentencia **DIM**, se pierde su contenido, adoptando la variable suscrita la nueva configuración indicada, y su contenido pasa a ser ceros si es numérica, y blancos si es de cadena.
Este efecto que «teóricamente» no es deseable, puede ser utilizado por nosotros mismos, a nuestra conveniencia, puesto que el «redimensionado» no producirá ninguna detención por error, sino simplemente el efecto descrito. En todo caso, lo habitual es no desear el borrado súbito del contenido de una matriz, debido a lo cual, deberemos tener cuidado con el lugar en que situamos las definiciones.



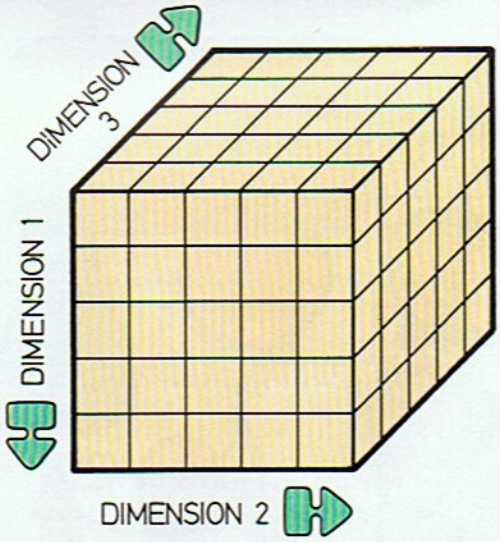
En las matrices numéricas, todos los elementos ocupan la misma memoria, independientemente de su contenido.

UN EJEMPLO

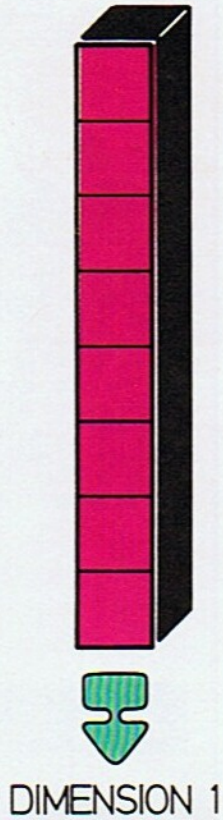
Para aclarar ideas, proponemos el siguiente programa de ejemplo, que nos pregunta aleatoria-



Las variables suscritas pueden alcanzar más de dos dimensiones, en cuyo caso se las suele conocer como MATRICES DE n DIMENSIONES.



Las matrices bidimensionales, se conocen como TABLAS.



Las variables suscritas unidimensionales, se denominan LISTAS o VECTORES.

¡!

La denominación general de las variables suscritas alfanuméricas (de cadena), es una sola letra seguida del símbolo dólar ($); lógicamente, el máximo número de variables de este tipo es también 26.

*

Para evitar confusiones entre índices y parámetros de *slicing* (troceado de cadenas), no es posible hacer coexistir en la memoria una variable suscrita de cadena y una cadena individual, con el mismo nombre.

*

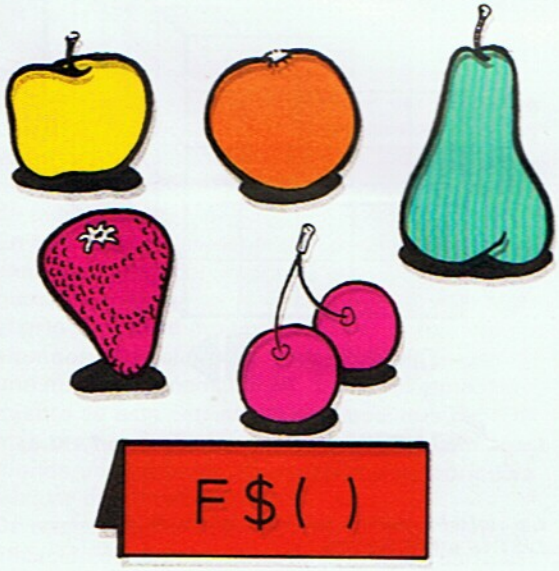
La definición inicial de las variables suscritas, se lleva a cabo mediante la sentencia **DIM**, en la cual se indican las características de la misma: nombre, y por tanto, tipo (numérica o alfanumérica), longitud y número de dimensiones.

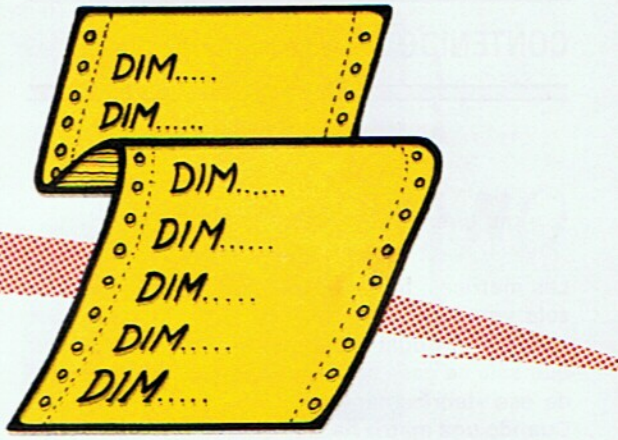
mente los nombres de 10 capitales europeas, doblando por cada acierto nuestra puntuación, y dejándola en la mitad a cada fallo, para alcanzar una evaluación final de los resultados:

```
  10 REM CAPITALES EUROPEAS - J.M.LOPEZ
MARTINEZ
  20 LET X$='                ': LET P=1
  30 DIM P$(22,32)
  40 FOR I=1 TO 22
  50 READ P$(I),P$(I,17 TO )
  60 NEXT I
  70 FOR I=1 TO 10
  80 LET X=INT (RND*22)+1: IF P$(X,32)='
#' THEN  GO TO 80
  90 LET P$(X,32)='#'
 100 INPUT (P$(X, TO 16)); LINE X$( TO 1
5)
 110 IF X$=P$(X,17 TO 31) THEN  LET E$='
CORRECTO!': LET P=2*P: GO TO 130
 120 LET E$='LO SIENTO': LET P=P/2
 130 PRINT E$,'ACUMULADO=';INT P''
 140 NEXT I
 150 IF P>=512 THEN  PRINT 'ENHORABUENA!
 POR LOS ';INT P;' PUNTOS': GO TO 170
 160 PRINT INT P;' PUNTOS. HAY QUE ESTUD
IAR MAS'
 170 STOP
 180 DATA 'ISLANDIA','REYKJAVIK','NORUEG
A','OSLO','SUECIA','ESTOCOLMO','FINLANDI
A','HELSINKI','U.R.S.S.','MOSCU'
 190 DATA 'DINAMARCA','COPENHAGUE','POLO
NIA','VARSOVIA','R.D.A.','BERLIN','R.F.A
.','BONN','AUSTRIA','VIENA'
 200 DATA 'CHECOSLOVAQUIA','PRAGA','IRLA
NDA','DUBLIN','REINO UNIDO','LONDRES','F
RANCIA','PARIS','SUIZA','BERNA'
 210 DATA 'HUNGRIA','BUDAPEST','RUMANIA'
,'BUCAREST','YUGOSLAVIA','BELGRADO','POR
TUGAL','LISBOA','ESPANA','MADRID'
 220 DATA 'ITALIA','ROMA','GRECIA','ATEN
AS'
```

Este programa tiene de todo. Entre otras cosas, algunos conceptos nuevos sobre temas que ya hemos comentado y aprovechamos ahora para ampliar.

Los conjuntos de variables alfanuméricas, se denominan mediante una sola letra seguida del símbolo dólar ($).





El número de **DIM**s *en un programa, así como su situación en el mismo, no tienen limitación alguna.*

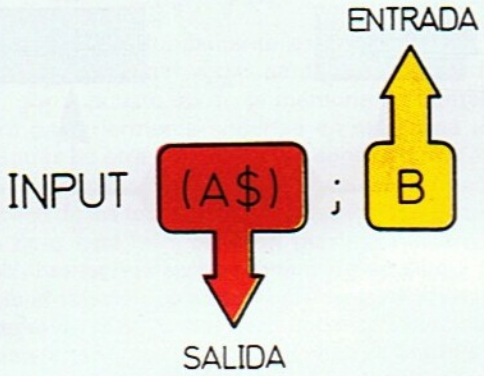
En el programa CAPITALES DE EUROPA, hemos utilizado el símbolo almohada (#) como «marca» que evita la repetición de preguntas.

Para la primera novedad, debemos fijarnos en las líneas 20 y 100. En la línea 20, asignamos el valor de 15 espacios a la variable **X$**. Esto es debido a que **X$** es la variable que empleamos para el **INPUT** de la capital en la línea 100, de la forma **X$ (TO 15)**. Generalizando, podemos conseguir variables de longitud fija utilizando este procedimiento: primero definir la variable con tantos blancos como deseemos, 15 en nuestro caso, y después hacer las posteriores asignaciones de valor por medio de **LET**, **READ** o **INPUT**, especificando qué parte de **X$**, en nuestro ejemplo **TO 15**.

Este modo de trabajar nos es muy útil en esta ocasión, puesto que, como sabemos, los elementos de las tablas alfanuméricas son todos de la misma longitud, es decir, se rellenan con blancos las posiciones no utilizadas. En este estado de cosas, la única manera de establecer comparaciones es este sistema. Para completar nuestro conocimiento sobre el comportamiento de los dimensionados alfanuméricos, hemos de saber que, al igual que las asignaciones de menor longitud se rellenan con blancos por la derecha, las más largas se truncan por la longitud máxima admitida.

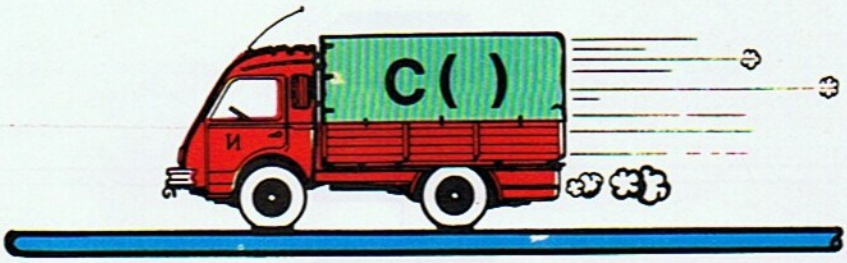
Así por ejemplo, si a un vector alfabético al que se le señaló una longitud máxima por elemento

de cinco caracteres, se le intenta asignar el valor "ANA", pasará a tener el contenido "ANA ". Por el contrario, si se le asigna "GLORIA", tomará el valor "GLORI".
En la línea 30, definimos el vector alfanumérico **P$(22,32)**, es decir, 22 elementos de 32 posiciones.
En las líneas 40 a 60, se establece la carga de la matriz, a partir de los datos contenidos en las líneas de data, 180 a 220. Como vemos, cada elemento de la matriz está dividido en dos mitades, de las cuales la primera (posiciones 1 a 16) contiene el país y la segunda (posiciones 17 a 32) contiene la capital.
En la línea 70 comienza el bucle de diez preguntas. Es evidente que precisamos una elección de preguntas al azar, pero sin que éstas puedan

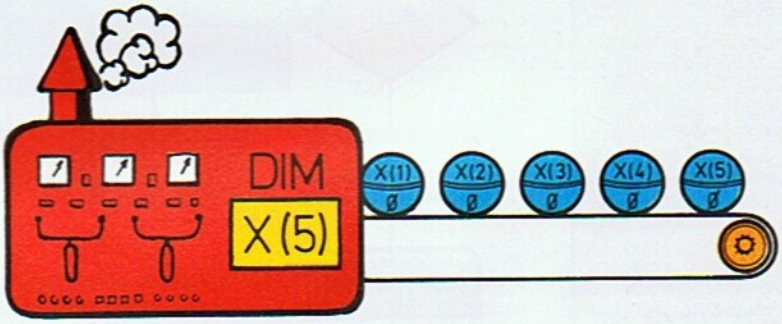


Cuando en un INPUT una variable aparece entre paréntesis, es considerada de salida (título).

repetirse. Esto se logra con las instrucciones 80 y 90. En 80, seleccionamos primero un número aleatorio entre 1 y 22 y comprobamos si tiene la marca de haber aparecido ya. Si la tiene, repetimos la generación aleatoria; si no, pasamos a la instrucción 90, donde marcamos el elemento de la tabla como utilizado, con un símbolo almohada (#) en la posición 32.
En la línea 100 utilizamos una nueva versión de **INPUT.** Es la modalidad de petición con texto variable, que consiste en indicar a continuación del **INPUT,** y entre paréntesis, el nombre de la variable que deseamos aparezca en la petición de los datos. Es muy importante que esta variable aparezca entre paréntesis, para distinguirla de la que se va a asignar en el **INPUT.**
En la línea 110, comparamos la respuesta tecleada (de longitud fija), con las posiciones 17 a 31 del elemento **(X)** seleccionado. En caso de acierto, se asigna a **E$** su valor, y se duplica el total acumulado de puntos, desviando la ejecución a la instrucción 130. A la línea 120 sólo se accede en caso de error, para colocar **E$** en su valor, y dejar el acumulado de puntos en la mitad.
En la línea 130 se imprime el valor de **E$,** evaluación de la respuesta, y el acumulado de puntos obtenidos hasta el momento, y en la línea 140 se cierra el bucle de preguntas abierto en la instrucción 70. Por último, las líneas 150 a 160, que sólo se alcanzan al final del ciclo de preguntas, evalúan los resultados obtenidos en base a la puntuación acumulada.
El programa en su estado actual, memoriza hasta 22 países europeos con sus respectivas capitales. Para ampliar sus conocimientos geográficos, sólo es necesario añadir las **DATA** correspondientes, teniendo en cuenta que también hemos de aumentar el número de elementos en el dimensionado de la matriz, y por supuesto, el bucle de lectura de las **DATA,** para su asignación a la variable suscrita. Del mismo modo, la extracción aleatoria de la pregunta también debe ser alterada, puesto que se mueve en el rango antes citado.

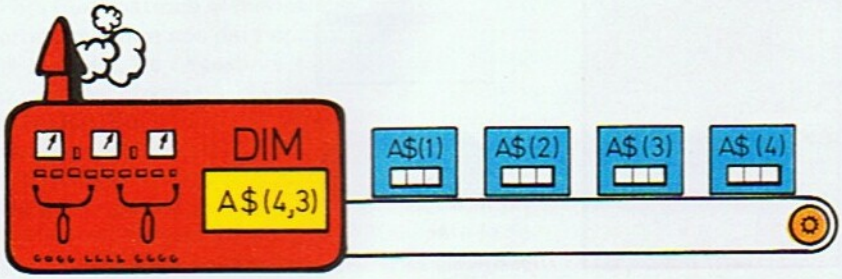


Una matriz numérica debe tener un nombre compuesto por una sola letra.



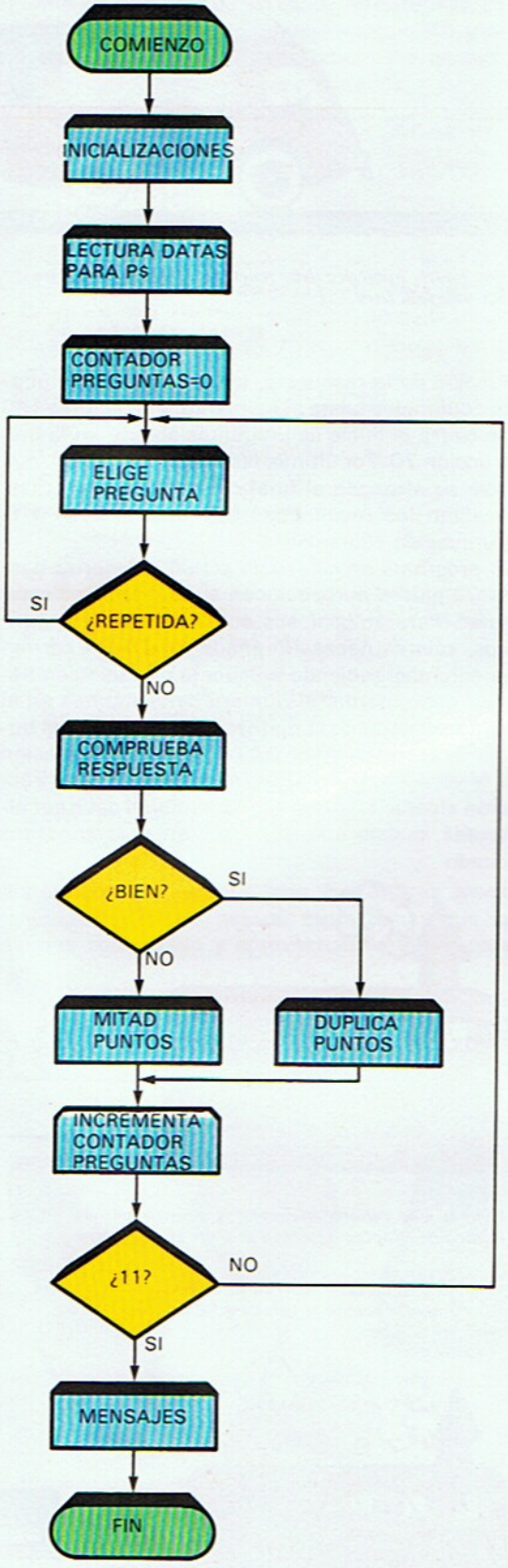
Cuando una variable numérica se dimensiona, sus elementos adoptan automáticamente el valor cero.

Cuando una variable suscrita de cadena se dimensiona, sus elementos toman el valor inicial de cadenas de espacios en blanco.

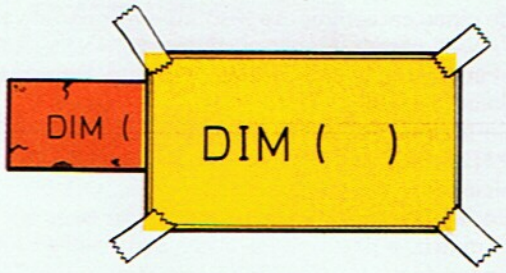


En las variables suscritas alfanuméricas, el parámetro que corresponde a la segunda dimensión en la sentencia **DIM**, se emplea para expresar la longitud máxima de los elementos de la variable.

Al contrario que las cadenas individuales, los elementos de una variable suscrita tienen una longitud fija, señalada en su correspondiente **DIM**.

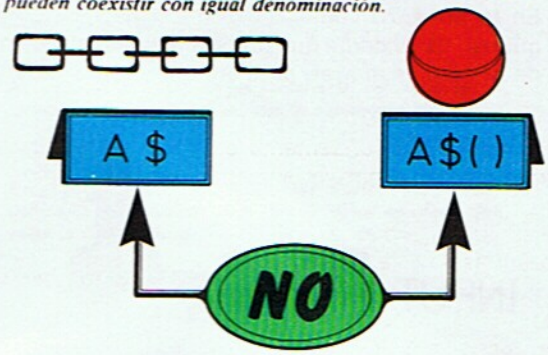


Organigrama del programa CAPITALES DE EUROPA.



Cualquier variable suscrita desaparece completamente al efectuarse un nuevo **DIM** *de la misma variable.*

Una variable de cadena y una suscrita del mismo tipo, no pueden coexistir con igual denominación.



Finalmente, para que la «filosofía» general del programa se conserve, hemos de respetar la distribución de los datos en el vector, es decir, la primera mitad de cada línea ha de contener el país, y la segunda, la respuesta.

Como demostración de lo fácil que es modificar este programa, añadiremos cinco preguntas más a nuestro test geográfico: las capitales de BULGARIA (SOFIA), ALBANIA (TIRANA), BELGICA (BRUSELAS), PAISES BAJOS (LA HAYA) y LUXEMBURGO (LUXEMBURGO). Siguiendo los pasos que hemos mencionado anteriormente, deberemos completar las líneas **DATA** que finalizan el programa con los nuevos datos:

```
230 DATA "BULGARIA","SOFIA","ALBANIA","TIRANA","PAISES BAJOS","LA HAYA","BELGICA","BRUSELAS","LUXEMBURGO","LUXEMBURGO"
```

Además, es necesario adaptar las dimensiones de la matriz, para que pueda albergar estos nuevos elementos, y por supuesto, prolongar el bucle de lecturas **(READ)**, puesto que el simple hecho de añadirlos a la **DATA** no implica que sean leídos.

```
30 DIM P$(27,32)
40 FOR I=1 TO 27
```

Por último, es preciso ampliar el rango de selección aleatoria de las preguntas, para que también sea posible escoger las recién llegadas:

```
80 LET X=INT (RND*27)+1:IF P$(X,32)="#" THEN GO TO 80
```

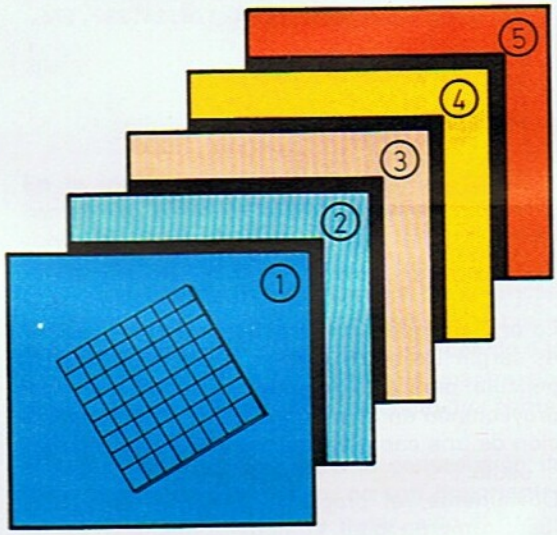
¡SE RUEDA!

EN este capítulo daremos ya por cerrado el tema del dinamismo en la pantalla completa, con una técnica basada en la subrutina de transferencia de memoria que vimos anteriormente. Lamentablemente, debido a la gran cantidad de memoria que se precisa, la técnica que a continuación exponemos está vedada para los usuarios del modelo de 16 K.

Realmente, su fundamento es muy sencillo. Se trata de almacenar varias pantallas en la memoria, por supuesto no en la memoria de pantalla, para posteriormente situarlas en esta zona una después de otra, de forma que consigan una animación similar a la que se obtiene en el cine con el paso de los fotogramas. La restricción en nuestro caso está bien clara: el máximo número de pantallas que se pueden almacenar son cinco, y con sólo cinco fotogramas mal vamos a hacer un largometraje.

El sistema de animación que ponemos en práctica en este capítulo, es similar al utilizado para la proyección de películas de cine.

No obstante, el efecto que se consigue con esta técnica es muy digno de tener en cuenta. El primer paso será obtener los necesarios cinco fotogramas; a tal fin, podemos introducir el programa de gráficos en tres dimensiones, aparecido en el capítulo segundo de la sección de programas. Una vez hecho esto, iremos creando pantallas de acuerdo con las fórmulas que a continuación se facilitan, y por el mismo orden. Cada pantalla deberá ser grabada una detrás de otra en una misma cinta, para que luego puedan ser utilizadas por el programa de demostración "CINEMA".



Gracias a la subrutina de transferencia en código máquina, la animación se consigue por superposición de imágenes a alta velocidad.

La grabación de las imágenes es muy fácil. Sólo hemos de esperar tranquilamente a que la figura correspondiente se dibuje, y cuando aparezca el mensaje **9 STOP statement, 470:1** podremos grabar la pantalla con el comando **SAVE "<nombre>"SCREEN$**. Lógicamente, el nombre para la grabación no tiene la menor importancia, y puede ser por ejemplo, el número de "fotograma" ("1", "2", "3", etc...).

Para que aparezca el mensaje de interrupción del programa, que nos dará opción a la grabación de la pantalla, es necesario, previamente a la ejecución del programa **3D**, introducir una pequeña modificación: cambiar la línea 470 para que quede como **470 STOP**.

Bien, ahora hemos de hacer el reparto de papeles para nuestra película: la función para la primera pantalla será **(X*X+Z*Z)/1000**, para la segunda pantalla utilizaremos **(X*X+Z*Z)/3000**,

BITS

Habitualmente, el color que el Spectrum utiliza como tinta para el marco de la pantalla, es el de contraste; si queremos utilizar otro color, debemos de indicarlo mediante un **POKE** a la Variable del Sistema **BORDCR** (23624). El valor a introducir será el código de color del marco multiplicado por ocho, más el código de tinta a emplear. Así por ejemplo, **POKE 23624,52** producirá la impresión en tinta verde sobre marco amarillo, puesto que 62 = 8 * 6 (código amarillo) + 4 (código verde). Este efecto desaparecerá al efectuar un nuevo **POKE** a esta dirección, o al ejecutar cualquier comando **BORDER**.

BITS

La lectura de información del casete no está sólo reservada al código máquina. Nosotros mismos, desde el BASIC, podemos averiguar mediante una sencilla operación, si el Spectrum está "escuchando" algún sonido. Tal y como se hace en el siguiente ejemplo, esta técnica puede emplearse para amenizar un programa hasta la carga del próximo, con la seguridad de que, sea cual sea el espacio entre ambos, no perderemos ni un bit de información.

```
10 BORDER 1: PAPER 1: CLS
20 FOR I=0 TO 6
30 BEEP .01,30+RND*2*I: BORDER 6-I
40 LET J=IN 254: IF J>127 THEN LET J=J-128
50 IF J>63 THEN LOAD ""
60 NEXT I: GO TO 20
```

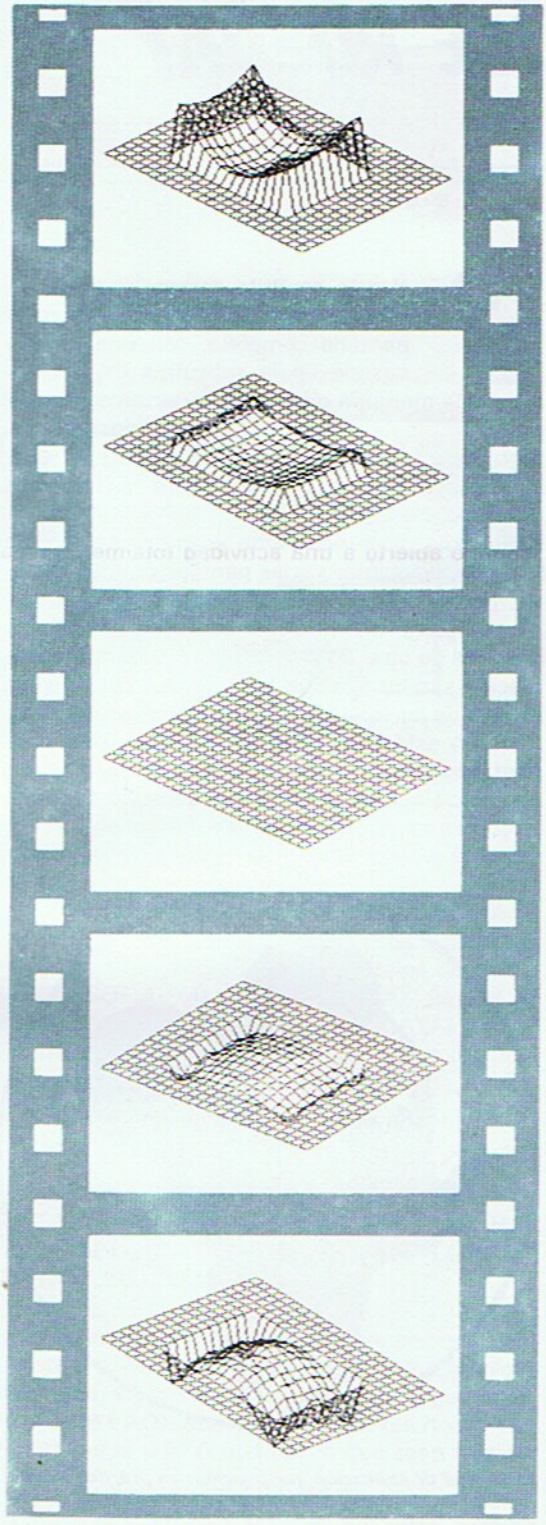
para la tercera **0** (dibujo plano), para el cuarto "fotograma" **SIN (((60-X*X-Z*Z) * (ABS SIN (60-X*X-Z*Z)-1))/5000)**, y para el último, **SIN (((60-X*X-Z*Z)*(ABS SIN (60-X*X-Z*Z) +1))/2000)**.

Una vez preparada la película, ya sólo queda proyectarla, para lo cual emplearemos el programa CINEMA.

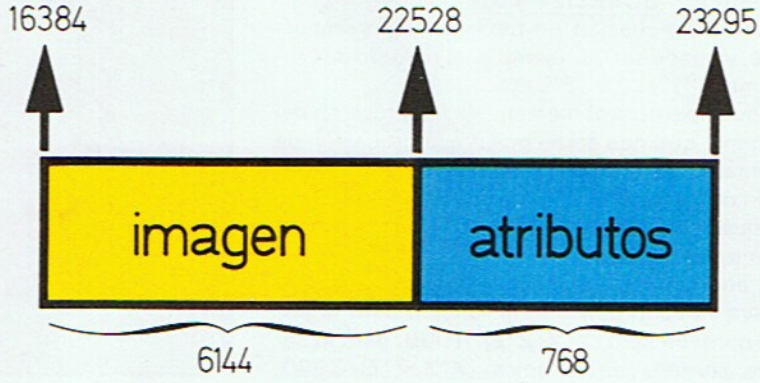
```
 10 REM CINEMA - C.DE LA OSSA & F.LOPEZ MARTINEZ
 20 CLEAR 34646
 30 LOAD "1"CODE 34647
 40 LOAD "2"CODE 40791
 50 LOAD "3"CODE 46935
 60 LOAD "4"CODE 53079
 70 LOAD "5"CODE 59223
 80 FOR I=23296 TO 23307
 90 READ A
100 POKE I,A
110 NEXT I
120 DATA 33,0,0,17,0,64,1,0,24,237,176,201
130 LET I=1
140 LET A=2
150 LET S=34647+A*6144
160 POKE 23298,INT (S/256)
170 POKE 23297,S-256*PEEK 23298
180 LET S=USR 23296
190 BEEP .1,-20+RND*60
200 LET A=A+I
210 IF A>4 OR A<0 THEN  LET I=-1*I: LET A=A+2*I
220 GO TO 150
```

Lo primero que hace el programa al ejecutarse, es cargar las cinco imágenes que configuran la película, para acto seguido, y cíclicamente, irlas proyectando en la pantalla, generando la animación de una cama elástica, o al menos "algo por el estilo".

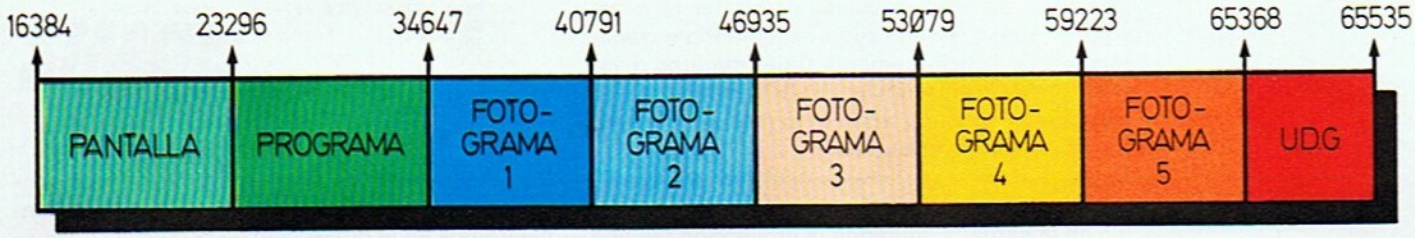
Obviamente, el programa CINEMA funcionará idénticamente igual, cualquiera que sean los fotogramas que se sitúen en la memoria; por tanto, podemos convertirnos en mini-George Lucas, y protagonizar la dirección de apasionantes secuencias a nuestro gusto: el estallido de una nave espacial, el movimiento de un brazo, el pestañeo de unos ojos, etc... En nuestro caso hemos empleado el programa **3D**, pues con un corto esfuerzo de introducción, nos permite disponer de los cinco fotogramas necesarios. Indudablemen-



Para hacer nuestra película, disponemos tan solo de cinco fotogramas.



Para ahorrar memoria, sólo se han almacenado los archivos de imagen, pero no las áreas de atributos; por tanto, nuestras películas son en blanco y negro.



En el gráfico se señala la distribución de memoria en el programa CINEMA.

te, otras secuencias considerablemente más artísticas pudieron ser obtenidas, sin embargo, su espectacularidad no justificaba la gran ocupación de espacio, y lo aburrido de su introducción. Como hemos visto, el programa CINEMA deja un camino abierto a una actividad totalmente creativa y artística, gracias a un ordenador. El punto más importante en este aspecto, es que los fotogramas estén lo suficientemente próximos en el tiempo, como para que no se produzcan saltos en la imagen, sino una armoniosa sensación de continuidad.

Sobre la subrutina de transferencia, sólo hay que añadir que es completamente reubicable, debido a lo cual, funcionará perfectamente cualquiera que sea la zona de memoria en que se almacene. Para ahorrar espacio, en el ejemplo ha sido depositada en el *buffer* (memoria intermedia) de la impresora. Por otra parte, el sistema de utilización de la rutina es muy simple: el primer código es siempre un 33, al cual le sigue la dirección en que se encuentra la pantalla a transferir, expresada en peso bajo-peso alto.

Los próximos tres códigos, no se cambian (17, 0 y 64), puesto que indican la dirección de destino de la transferencia, la cual, lógicamente, es siempre la misma: la pantalla (16384). Por último, al próximo código (1), le debe seguir, expresado nuevamente en bajo-alto, el número de bytes a transferir; en nuestro caso son 6144, es decir, el archivo de imagen, y por tanto, los bytes codificados son 0 y 24 (24*256+0=6144). Si quisieramos trasladar la pantalla completa, incluyendo el área de atributos, los bytes a codificar serían 0 y 27, correspondientes a los 6912 bytes que integran la pantalla completa (27*256+0=6912).

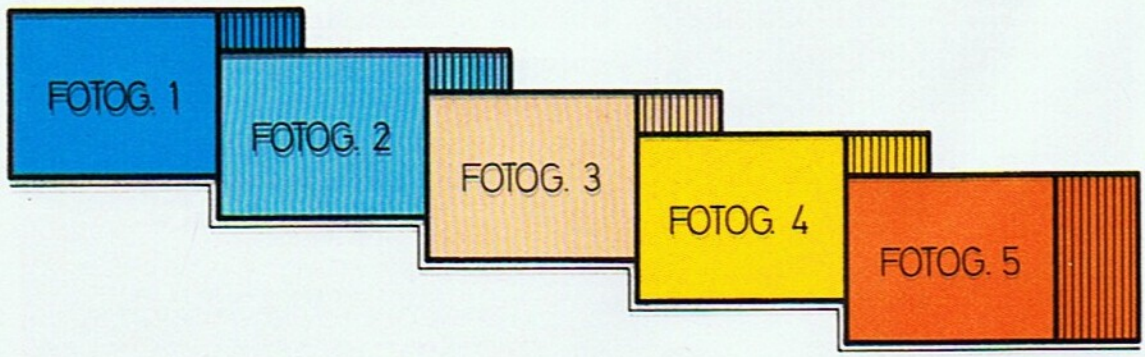
CINEMASCOPE

En la introducción al dinamismo de pantallas completas, comentamos la posibilidad de, utilizando las mismas técnicas, poder animar una determinada zona de la pantalla, de fácil tratamiento por la subrutina de transferencia. Ya entonces, sugerimos para la creación de estos efectos la utilización de tercios de pantalla, que debido a la peculiar organización del archivo de imagen, podían ser tratados prácticamente como una pantalla independiente.

Como vimos entonces, para la consecución de esta técnica de dinamismo, no son importantes las dimensiones del área a tratar, sino el hecho de que las imágenes sean transferidas completas, es decir, sin tomar en consideración qué fi-

Por motivos de memoria, la técnica descrita en este capítulo, está vedada a los usuarios del modelo de 16 K.

Debido a las direcciones de almacenamiento, aunque las pantallas se cargan completas, sus respectivos archivos de imagen solapan el área de atributos de la anterior.



guras son de fondo y cuales de primer término, y por supuesto, que la zona de memoria a transferir sea lo suficientemente uniforme (uno o varios tercios consecutivos de la pantalla), como para que admita un tratamiento mediante la subrutina en código máquina.

Nuestro último experimento será realizar la proyección de un "cortísimometraje" en el tercio medio de la pantalla, imitando la proyección del tipo CINEMASCOPE. Como en el caso anterior, la confección del "film", consta de dos fases. En primer lugar, un árduo rodaje, cuyo protagonista será el conocido actor TORPEDO RUN, y que será llevado a cabo mediante el programa ACCION!

Ha nacido una estrella: TORPEDO RUN.

```
  10 REM ACCION! - C.DE LA OSSA & F.LOPE
Z MARTINEZ
  20 FOR I=0 TO 7: READ A: POKE USR "D"+
I,A: NEXT I: DATA 56,24,24,48,96,103,126
,56
  30 PRINT AT 12,15;"D": PLOT 123,80: DR
AW 0,32
  40 LET I=32: GO SUB 3000
  50 FOR I=30 TO 18 STEP -2
  60 LET F=31: GO SUB 2000
  70 GO SUB 3000
  80 NEXT I: CLS
  90 FOR I=16 TO 2 STEP -2
 100 LET F=I+13: GO SUB 2000
 110 PRINT AT 12,I-1;"D": PLOT (I-1)*8+4
,80: DRAW -(I-16)*8,32
 120 GO SUB 4000
 130 GO SUB 3000
 140 CLS
 150 NEXT I
 160 LET I=0: LET F=13: GO SUB 2000
 170 PRINT AT 9,10;"D": PLOT 84,104: DRA
W 48,7
 180 GO SUB 4000: GO SUB 3000
 190 CLS
 200 GO SUB 2000
 210 PRINT AT 12,19;"4337 3374  7"
 220 PRINT AT 13,19;"537   5 56 5"
 230 PRINT AT 14,19;"5     5 5 65"
 240 PRINT AT 15,19;"1    3321  2"
 250 PLOT 56,88: DRAW 7,7: DRAW 0,-7: PL
OT 56,63: DRAW 7,-7: DRAW 0,7
 260 LET I=-2: GO SUB 3000
 270 GO TO 10000
1000 DATA 0,0,0,0,1,2,4,4,24,40,72,72,72
,40,24,4,4,2,1,0,0,0,0,0,31,48,64,128,0,
0,0,0
1010 DATA 0,0,0,0,0,0,0,0,0,0,0,128,64,3
2,24,15,255,17,33,65,178,244,248,112,0,0
,0,0,0,0,1
1020 DATA 1,1,6,9,9,50,194,255,255,0,0,0
,0,0,0,0,0,0,128,128,128,128,128,128,1
28,128,0,0,0,0,255
1030 DATA 255,15,15,15,240,240,240,240,1
5,15,15,15,240,240,240,240,15,15,15,15,2
40,240,240,255
1040 DATA 255,0,0,15,15,0,15,15,0,15,15,
0,15,15,0,15,15,0,15,15,0,0,0,255
1050 DATA 255,0,0,255,255,0,255,131,0,13
1,131,0,131,255,0,255,255,0,131,131,0,0,
0,255,255,0,0,131,195,0,227,227,0,227,22
7,0,227,195,0,131
1060 DATA 195,0,225,224,0,0,0,255,255,0,
0,224,224,0,224,224,0,224,224,0,224,224,
0,224,255,0,255,255,0,0,0,255
1070 DATA 255,0,0,248,248,0,248,248,0,24
8,248,0,248,248,0,248,248,0,240,224,0,0,
0,255,255,0,0,63,127,0,255,248
1080 DATA 0,248,248,0,248,248,0,248,248,
0,248,248,0,0,0,255,255,0,0,248,252,0,25
4,62,0,62,62,0,62,62,0,62
1090 DATA 62,0,62,62,0,0,0,255,128,64,32
,16,8,4,2,1,0,0,0,0,0,0,0,0,1,2,4,8,16,3
2,64,128
1100 DATA 255,129,129,129,129,129,129,12
9,129,129,129,129,129,129,129,129,129,12
9,129,129,129,129,129,255
2000 RESTORE 1000
2010 FOR J=I TO F
2020 FOR K=0 TO 2
2030 FOR L=0 TO 7: READ A: POKE USR "A"+
L+K*8,A: NEXT L
2040 PRINT AT 11+K,J;CHR$ (144+K)
2050 NEXT K
2060 NEXT J
2070 RETURN
3000 LET N$=STR$ (17-I/2): SAVE N$CODE 1
8432,2048
3010 RETURN
4000 FOR J=0 TO 39: PLOT (I+14)*8+RND*15
,72+8*RND: NEXT J: PLOT (I+7)*8,88: DRAW
 7,7: DRAW 0,-7: PLOT (I+7)*8,63: DRAW 7
,-7: DRAW 0,7
4010 RETURN
```

Con la ejecución de este programa, se irán conformando las diferentes imágenes que luego emplearemos en la proyección de la película (segunda fase). Por cada una de ellas, aparecerá en la parte inferior de la pantalla el mensaje habitual de grabación *Start tape then press any key;* de esta forma iremos grabando en una cinta, una a continuación de otra, las dieciocho pantallas diferentes que se nos presenten. En esta ocasión, el número de fotogramas ha aumentado sensiblemente, debido a que la cantidad de memoria que utiliza cada uno es una tercera parte de la pantalla completa (2048 bytes).

Finalmente, el programa TORPEDO, cuyo listado aparece al final de este artículo, pondrá en movimiento las imágenes "rodadas" con TORPEDO RUN. Para ello, se sigue un sistema prácticamente idéntico al del programa CINEMA, con la diferencia de que en esta ocasión se realiza la carga de dieciocho fotogramas, y por lo tanto, no en las mismas posiciones que en el ejemplo a pantalla completa. Por otra parte, la rutina de transferencia ha sido ligeramente alterada, puesto que esta vez, varía tanto la zona de destino de la imagen, como el número de bytes a transferir. La dirección de destino es ahora el primer byte del segundo tercio de la pantalla (16384 + 2048 = 18432). Del mismo modo, el número de bytes a transferir es sólo un tercio (2 K), lo que suponen 2048 bytes.

```
  10 REM TORPEDO - C.DE LA OSSA & F.LOPE
Z MARTINEZ
  20 CLEAR 28499
  30 BORDER 0: PAPER 0: INK 7: CLS : PRI
NT AT 11,6;"VISITE NUESTRO BAR"
  40 FOR I=1 TO 18
  50 LOAD (STR$ I)CODE 28500+2048*(I-1),
2048
  60 IF I>14 THEN  BEEP .5,30
  70 NEXT I
  80 FOR I=23296 TO 23307
  90 READ A: POKE I,A: NEXT I
 100 DATA 33,0,0,17,0,72,1,0,8,237,176,2
01
 110 PRINT AT 21,0;"CINEMASCOPE"
 120 PAPER 2: INK 7: GO SUB 1000
 130 FOR I=8 TO 15: READ A$: PRINT AT I,
0;A$: NEXT I
 140 DATA "              CAST","UN FILM D
E INGELEK/NUEVA LENTE","PRODUCTOR: VICEN
TE ROBLES","DIRECTOR: FERNANDO LOPEZ","R
EALIZADOR: CARLOS DE LA OSSA","AYTE.DIRE
CCION: PILAR MANZANERA","FOTOGRAFIA: EQU
IPO GALATA","MAQUILLAJE: CARLOS GLEZ.-AM
EZUA"
 150 BEEP 10,0
 160 PAPER 5: INK 0: GO SUB 1000
 170 FOR I=1 TO 18
 180 LET S=28500+2048*(I-1)
 190 LET SA=INT (S/256): LET SB=S-256*SA
 200 POKE 23297,SB: POKE 23298,SA
 210 RANDOMIZE USR 23296
 220 PAUSE 1
 230 NEXT I
 240 BEEP .1,30: PAUSE 0: GO TO 170
1000 PRINT AT 8,0;: FOR I=0 TO 7: PRINT
"                              ": NEXT
 I: RETURN
```

SIMON

ESTE nuevo programa nos ayudará a ejercitar nuestra capacidad de retención, tanto visual como auditiva, al mismo tiempo que nos divertiremos y pasaremos un rato muy agradable.

El juego del SIMON es una competencia contra nosotros mismos, es decir, un solitario, en el cual la máquina tiene un papel pasivo, de forma similar al MASTER MIND (MASTER CUB, para los lectores de esta Obra).

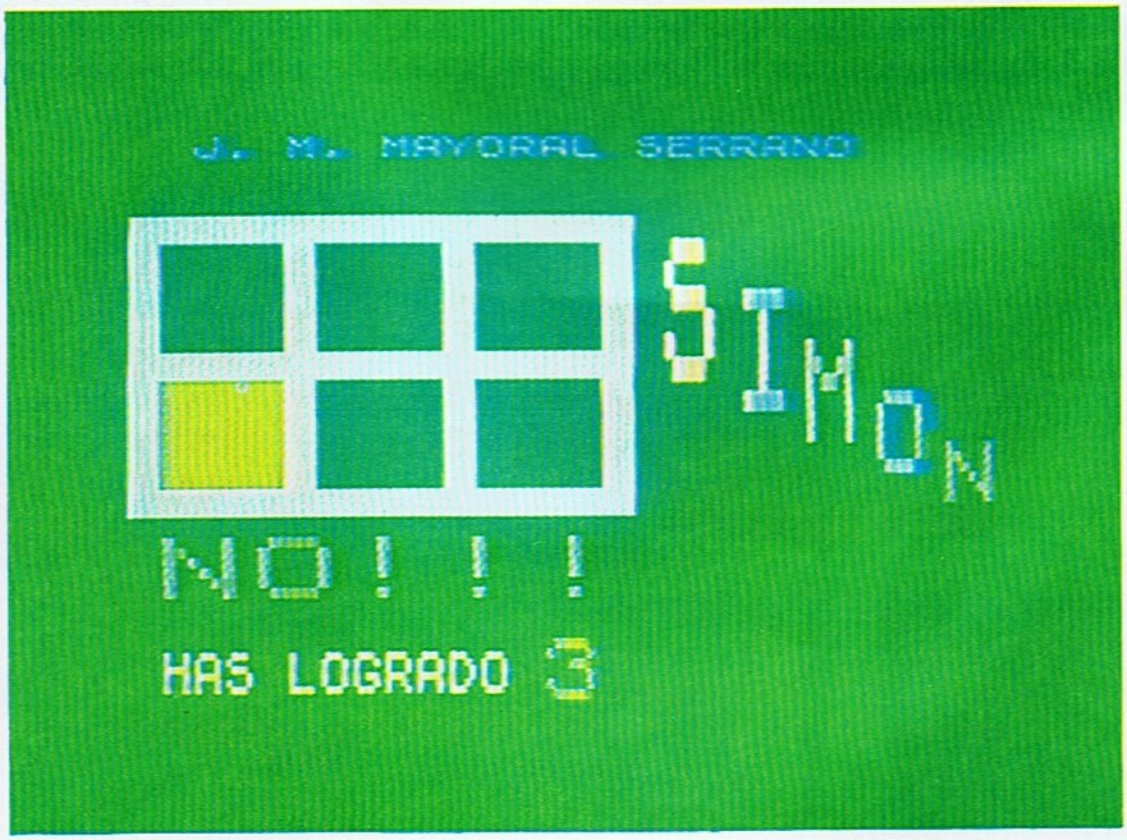
El ordenador emite un determinado sonido, al tiempo que presenta un cuadrado de color en la pantalla; a continuación debemos pulsar la tecla correspondiente a ese color, o si lo preferimos, a ese sonido. Si conseguimos acertarlo, al sonido y color anteriores se añadirá uno nuevo, diferente o no. Esta vez debemos repetir los dos colores. El juego continuará aumentando la secuencia, hasta que nos equivoquemos en alguna de ellas.

Nuestro programa posee dos niveles de dificultad. El primero y más sencillo, hace uso de los cuatro primeros colores básicos del Spectrum: azul, rojo, magenta y verde. El nivel restante, utiliza dos colores más, siguiendo el orden antes expuesto: azul, rojo, magenta, verde, cián y amarillo.

Estos colores son generados aleatoriamente uno a uno, y almacenados en una tabla para su posterior presentación en pantalla. Por cada secuencia de colores acertada, nuestro ordenador elige otro, añadiéndole al final de la cadena de colores, que va aumentando a medida que avanzamos en el juego.

Por cada color generado, se emite siempre un mismo sonido, de forma que si no somos capaces de recordar la secuencia policromada presentada en la pantalla, nuestro Spectrum nos da otra

La presentación del programa en la pantalla, gana mucho en calidad con la utilización de la subrutina de caracteres gigantes.



¡!

El programa carece de gráficos definidos, debido a lo cual en el listado no aparecen caracteres subrayados.

*

Para grabar el programa, utilizaremos la acción combinada de los comandos **SAVE** y **LINE** de la siguiente forma:
SAVE ''SIMON'' LINE 1480.

opción: recordarla dependiendo de la secuencia de notas que «deja» escapar. En el momento en que fallemos un color o nota, el Spectrum nos lo hará saber inmediatamente, a la vez que nos informará de los puntos obtenidos, finalizando el juego en ese momento.
La forma de puntuar de nuestro Spectrum es la siguiente: sumará al marcador tantos puntos como notas o colores tenga la secuencia presentada, siempre y cuando se acierte dicha secuencia. Es decir, si el ordenador ha generado 7 colores y hemos sido capaces de recordarlos por su orden cronológico de aparición, sumará 7 al marcador, mientras que si fallamos en alguno de ellos, la puntuación en esa secuencia será 0 y la partida habrá finalizado.

EL PROGRAMA

Este programa puede almacenar una serie de 200 colores con sus notas correspondientes. Si queremos aumentar o disminuir dicha cantidad, no tenemos más que variar el número que define la longitud de la matriz S, que lógicamente está inicializada en 200 elementos: **DIM S(200)** (línea 130).

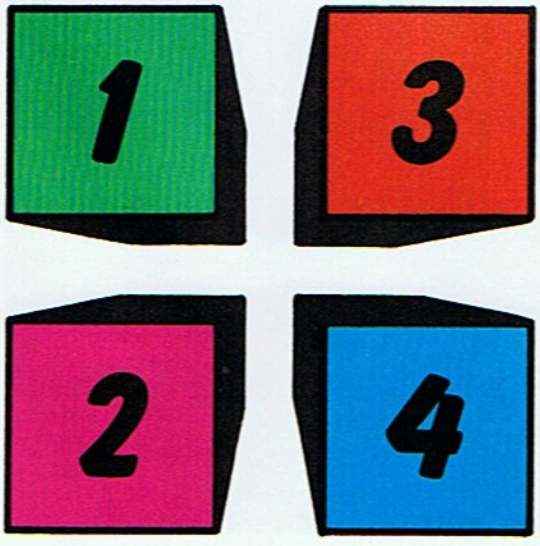
¡!

Los dos niveles de dificultad que posee el juego, hacen referencia al número de colores que pueden entrar en la secuencia a recordar (cuatro o seis).

*

Las teclas que hay que pulsar para emitir la secuencia de colores y sonidos, son las de los números correspondientes a los códigos de color deseados.

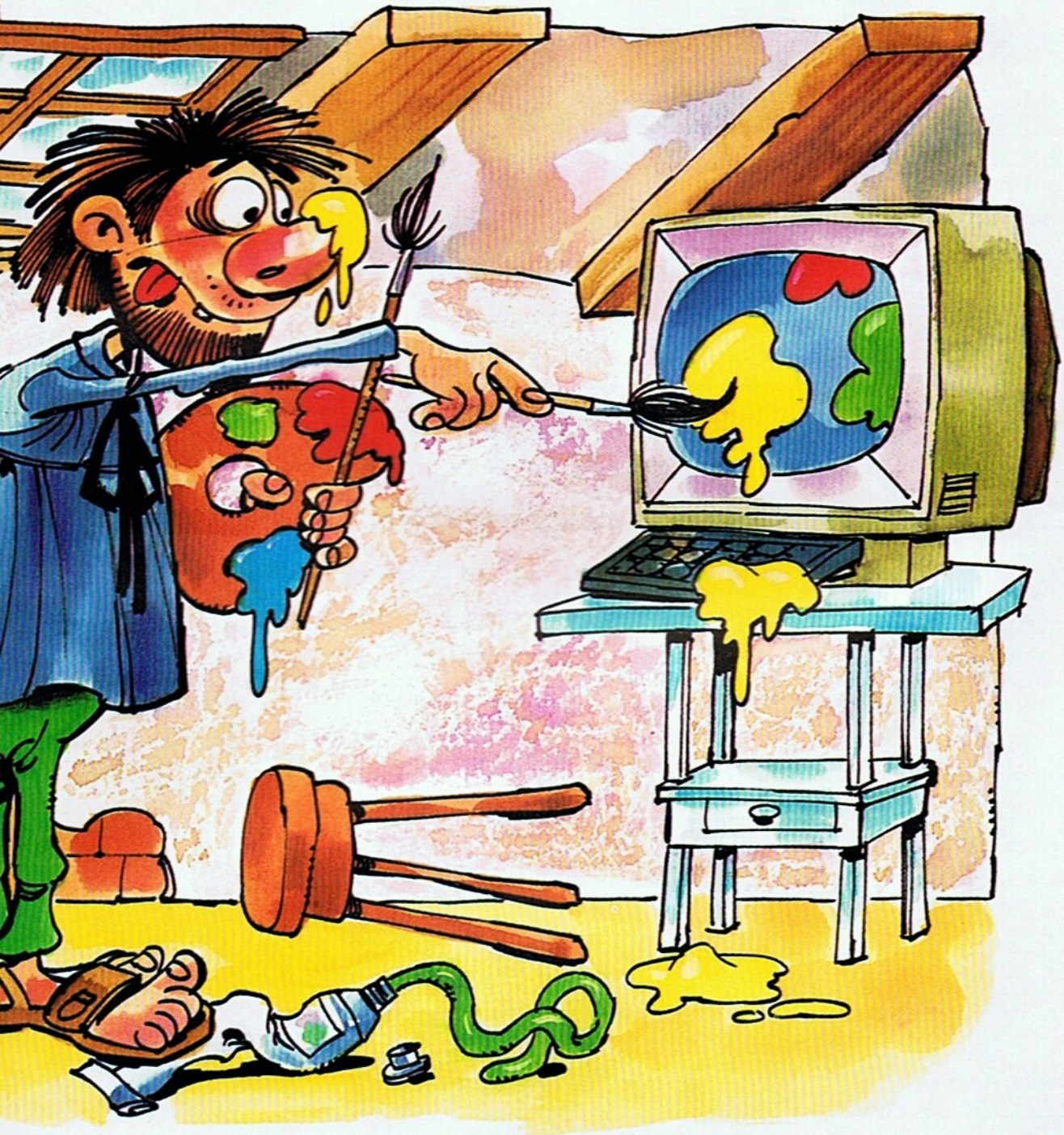
El nivel más sencillo de nuestro juego del SIMON, se basa en los cuatro primeros colores del Spectrum.



El tiempo que un color está presente en la pantalla, depende del nivel de dificultad, y se controla mediante la variable **T**, que se utiliza como parámetro del comando **BEEP**. Esta variable numérica, según el nivel de dificultad elegido, puede tener dos valores: en el nivel uno, **T** adquiere el valor 0.5, y en el segundo nivel su valor es 0.05. De esta forma, cuanto más pequeño sea **T**, menos tiempo estará el color generado en la pantalla.

En SIMON volvemos a utilizar la subrutina de caracteres gigantes de PSION, cuidando así la estética del programa en la pantalla. Por tanto, seguiremos para la adopción de la rutina las especificaciones que a este respecto se han brindado en programas anteriores, que también hayan hecho uso de ella.

En esta ocasión, echaremos de menos los caracteres subrayados en el listado, puesto que no se emplean gráficos, sino diferentes códigos de **PAPER** para la generación de los colores.
Es importante recordar la absoluta necesidad de tener cargada la subrutina de caracteres gigantes para el correcto funcionamiento del programa; así pues, para almacenarlo, utilizaremos el siguiente comando: **SAVE "SIMON" LINE 1480.** Disponiendo la autoejecución en la línea 1480, que es la encargada de la lectura del código máquina.



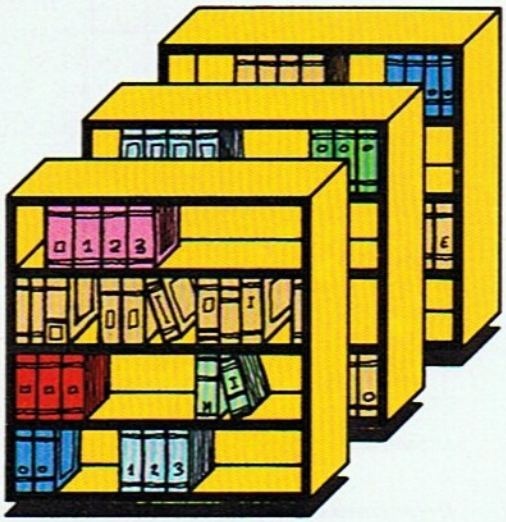
```
  10 REM ************************
  20 REM * J.M.MAYORAL SERRANO *
  30 REM ************************
  40 REM *   Program. SIMON    *
  50 REM ************************
  60 REM INICI. - VARIABLES
  70 BORDER 0: PAPER 0: INK 9: CLS : POKE 23658,8
  80 LET XX=140
  90 LET YY=8
 100 LET C=10
 110 DIM A$(21)
 120 LET N1=0
 130 DIM S(200): LET IND=1
 140 GO SUB 1300
 150 REM MENU DE PRESENTACION
 160 PRINT AT 10,3; FLASH 1; INK 6; PAPER 2;1
 170 PRINT AT 10,7;"~SIN COMPLICACIONES~"
 180 PRINT AT 12,3; FLASH 1; INK 2; PAPER 6;2
 190 PRINT AT 12,7;"~CON BASTANTES PROBLEMAS~"
 200 PRINT AT 5,7; PAPER 5; INK 1;"NIVEL DE DIFICULTA
D"
 210 PRINT AT 21,1;"# # # #  ELIGE OPCION  # # # #"
 220 LET K$=INKEY$
 230 IF CODE K$>50 OR CODE K$<49 THEN  GO SUB 260: GO
TO 220
 240 BEEP .2,40
 250 GO TO 350
 260 REM AMENIZAR LINEA 210
 270 IF C=23 THEN  LET C=10
 280 PRINT  INK 4; OVER 1;AT 21,C;"  "
 290 BEEP .01,40
 300 PRINT  INK 7; OVER 1;AT 21,C-1;" "
 310 BEEP .01,45
 320 LET C=C+1
 330 RETURN
 340 REM NUM. DE COLORES
 350 LET CLRS=2*(VAL K$+1)
 360 IF CLRS=4 THEN  LET T=.5: GO TO 380
 370 LET T=.05
 380 CLS
 390 REM NUM. SONI. COMIENZO
 400 PRINT AT 19,0;"CON CUANTOS TONOS DESEAS EMPEZAR"
 410 PRINT AT 21,12;"1 - 5"
 420 LET K$=INKEY$
 430 IF CODE K$<49 OR CODE K$>53 THEN  GO TO 420
 440 BEEP .2,40: CLS
 450 LET CONT=VAL K$
 460 PRINT AT 0,5; INK 3;A$
 470 REM DIBUJO MARGEN
 480 IF CLRS=6 THEN  GO TO 590
 490 FOR N=3 TO 13 STEP 5
 500 PRINT  PAPER 7;AT N,3;"                    "
 510 NEXT N
 520 PRINT AT 0,5; INK 3;A$
 530 FOR W=3 TO 13
 540 FOR N=3 TO 13 STEP 5
 550 PRINT  PAPER 7;AT W,N;" "
 560 NEXT N
 570 NEXT W
 580 GO TO 700
 590 FOR N=3 TO 13 STEP 5
 600 PRINT AT 0,5; INK 3;A$
 610 PRINT  PAPER 7;AT N,3;"                    "
 620 NEXT N
 630 FOR W=3 TO 13
 640 FOR N=3 TO 18 STEP 5
 650 PRINT  PAPER 7;AT W,N;" "
 660 NEXT N
 670 PRINT AT 0,5; INK 3;A$
 680 NEXT W
 690 DATA "S",2,7,"I",2,6,"M",2,5,"O",2,4,"N",2,3
 700 RESTORE 690
 710 FOR G=1 TO 5
 720 LET XX=XX+18
 730 LET YY=YY+16
 740 READ P$,XS,YS:
 750 BRIGHT 1: INK YS-1
 760 GO SUB 1390
 770 NEXT G
 780 INK 9: BRIGHT 0
 790 REM PROGRAMA PRINCIPAL
 800 FOR N=1 TO CONT-1
 810 LET S(IND)=INT (RND*CLRS)+1
 820 LET IND=IND+1
 830 NEXT N
 840 REM COMIENZO SONIDOS
 850 PRINT AT 0,5; INK 3;A$
 860 LET S(IND)=INT (RND*CLRS)+1
 870 FOR N=1 TO IND
 880 LET K$=STR$ S(N): GO SUB 1150
 890 BEEP T,S(N)*7: GO SUB 1290
 900 NEXT N
 910 LET IND=IND+1
 920 REM INTENTOS ADVERSARIO
 930 FOR N=1 TO IND-1
 940 LET K$=INKEY$
 950 IF CODE K$<49 OR CODE K$>CODE STR$ CLRS THEN  GO
TO 940
 960 IF VAL K$=S(N) THEN  GO SUB 1150: BEEP .6,S(N)*7
: GO SUB 1280: GO TO 980
 970 GO TO 1070
 980 NEXT N
 990 LET N1=N1+N-1
1000 PRINT AT 18,10;"MUY BIEN"
1010 PRINT AT 21,10;"CONTINUO"
1020 FOR P=1 TO 7: BEEP .1,35: NEXT P
1030 PRINT AT 18,10;"        "
1040 PRINT AT 21,10;"        "
1050 GO TO 840
1060 REM FALLOS
1070 GO SUB 1150: BEEP 1,-10: LET P$="NO!!!": LET YY=
115: LET XX=30: LET XS=3: LET YS=3: INK 4: GO SUB 139
0:
1080 LET P$="HAS LOGRADO": LET XX=32: LET YY=150: LET
 XS=1: LET YS=2: INK 6: GO SUB 1390
1090 LET P$=STR$ N1: LET XX=127: LET XS=2: LET YS=3:
LET YY=145: INK 2: GO SUB 1390
1100 LET XX=LEN P$*23+127: LET P$="PUNTOS": LET YY=15
0: LET XS=1: LET YS=2: INK 6: GO SUB 1390
1110 LET P$="OTRA PARTIDITA ?": LET YY=175: LET XS=2:
 LET YS=1: INK 5: GO SUB 1380
1120 IF INKEY$="S" THEN  RUN
1130 IF INKEY$<>"N" THEN  GO TO 1120
1140 GO TO 10000
1150 REM PINTOR DE CUADROS
1160 LET Q=VAL K$
1170 IF Q=1 THEN  LET F=4: LET CO=4: GO TO 1240
1180 IF Q=3 THEN  LET F=4: LET CO=9: GO TO 1240
1190 IF Q=5 THEN  LET F=4: LET CO=14: GO TO 1240
1200 IF Q=2 THEN  LET F=9: LET CO=4: GO TO 1240
1210 IF Q=4 THEN  LET F=9: LET CO=9: GO TO 1240
1220 IF Q=6 THEN  LET F=9: LET CO=14: GO TO 1240
1230 PRINT "ERROR": STOP
1240 FOR P=F TO F+3
1250 PRINT  PAPER Q;AT P,CO;"    "
1260 NEXT P
1270 RETURN
1280 REM BORRADO
1290 LET Q=0: GO TO 1240
1300 REM : REM : REM
1310 RESTORE 1320
1320 DATA 74,46,32,77,46,32,77,65,89,79,82,65,76,32,8
3,69,82,82,65,78,79
1330 FOR N=1 TO 21
1340 READ A: LET A$(N)=CHR$ A
1350 NEXT N
1360 RETURN
1370 REM BASIC CARACT. GIGANTES
1380 LET xx=(256-8*xs*LEN p$)/2
1390 LET i=23306
1400 POKE i,xx: POKE i+1,yy: POKE i+2,xs: POKE i+3,ys
: POKE i+4,8
1410 LET i=i+4: LET w=LEN p$
1420 FOR q=1 TO w
1430 POKE i+q,CODE p$(q)
1440 NEXT q
1450 POKE i+w+1,255
1460 LET w=USR 32256
1470 RETURN
1480 REM  AUTO-EJEC. OBLIGATORIA
1490 CLEAR 32255
1500 LOAD "LIT"CODE
1510 RUN
```

PRACTICAS CON MATRICES

SEGURAMENTE lo más complicado del manejo de matrices sea su aplicación práctica. La verdad es que, con los conocimientos adquiridos hasta ahora, podemos emplear de forma eficaz los vectores y tablas en nuestros programas. Sin embargo, para tener una mayor orientación sobre cómo manejar esta potente herramienta de programación, propondremos algunos ejemplos prácticos, tratando de abarcar, al menos, las aplicaciones más inmediatas de la teoría.

VECTORES

Los vectores o matrices unidimensionales, son la expresión más simple de una estructura de conjunto. Nos encontramos pues, ante una serie ordenada de un determinado número de elementos, calificados por un índice. Decimos que la serie es ordenada porque, con independencia del contenido y la estructura misma de los elementos que la componen, podemos conocer la posición relativa de cualquiera de ellos dentro de la serie, así como cual le precede y cual le sigue.



J$ (8,16)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	J	U	A	N		M	A	N	U	E	L					
2	J	O	S	E		F	E	R	N	A	N	D	O			
3	V	I	C	E	N	T	E									
4	C	A	R	L	O	S										
5	A	N	A													
6	J	O	S	E												
7	M	I	L	A	G	R	O	S								
8	A	N	T	O	N	I	O									

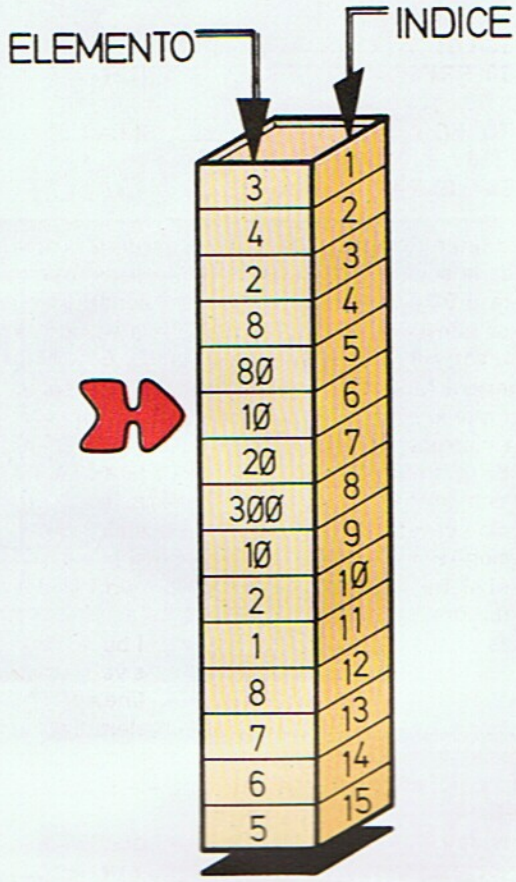
La representación de un vector de cadena es una tabla.

Supongamos, por ejemplo, un juego en el que intervengan **N** jugadores. Si almacenamos sus nombres en una matriz, siempre sabremos a qué jugador le corresponde el siguiente turno de juego, sólo con sumar 1 al índice del vector, y controlando el hecho de hacer éste igual a 1 cuando, por efecto de la operación de suma, su valor se haga igual a **N+1**, es decir, exceda al último jugador:

```
 10 REM - TURNO DE JUGADORES
 20 INPUT "Numero de jugadores:";N
 30 DIM J$(N,16)
 40 FOR I=1 TO N
 50 INPUT "Jugador  "+STR$ I+":";LINE
J$(I)
 60 NEXT I
 70 PRINT "TURNOS","JUGADORES"
 80 FOR I=0 TO 31: PRINT "=";: NEXT I
 90 PRINT
100 FOR I=1 TO N: PRINT I,J$(I): NEXT I
```

Las matrices aportan al BASIC un sistema altamente eficaz para el almacenamiento y clasificación de datos.

Una matriz es una serie ordenada, en la cual, independientemente de su contenido, podemos conocer mediante el índice, la posición relativa que ocupan sus elementos.



¡!

La hipotética cuarta dimensión, tan traída y llevada por la ciencia ficción, puede ser claramente imaginada informáticamente mediante una estructura tetradimensional. Su representación geométrica es evidentemente bastante compleja, aunque generalmente se le asigna el sentido de tiempo a la cuarta dimensión, de forma que estas matrices pueden representarse mediante estructuras espaciales (tridimensionales), analizadas en un determinado instante (cuarta dimensión).

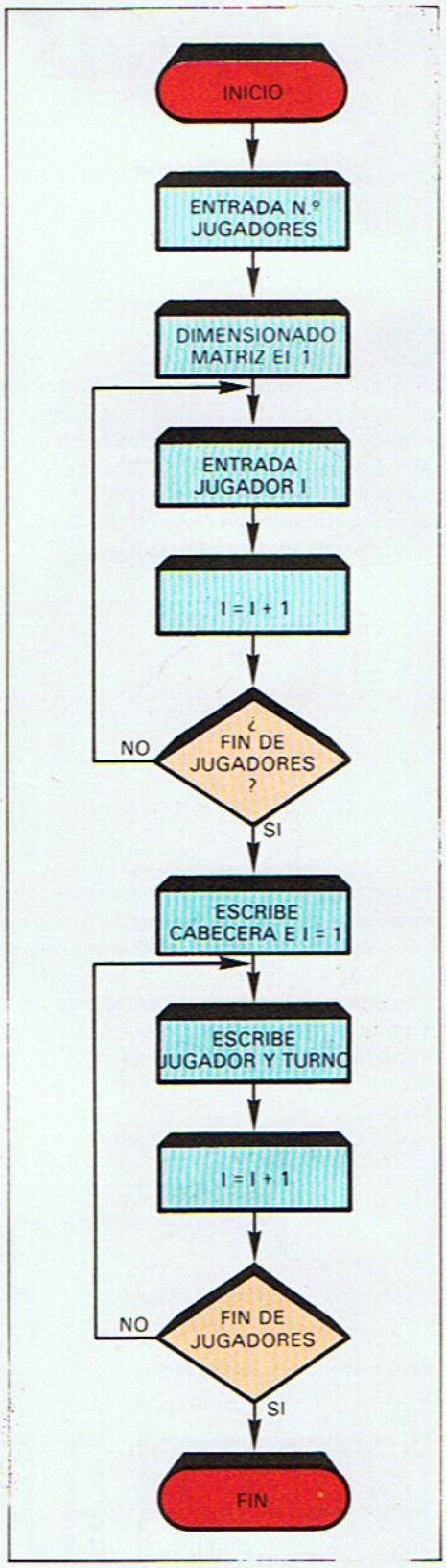
*

El vector es una serie ordenada porque, con independencia del contenido de los elementos que lo componen, sabemos la posición relativa de uno determinado de ellos dentro de la serie, facilitándonos así mismo conocer cual es el que le precede y el que le sigue.

MATRICES NUMERICAS DE UNA DIMENSION

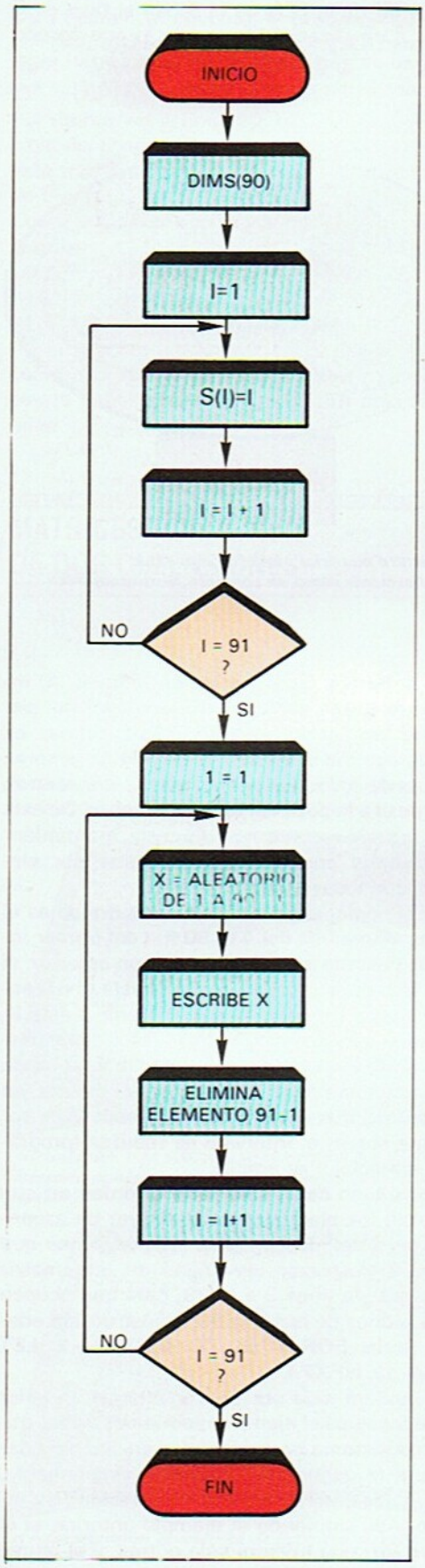
Las matrices pueden definirse como multidimensionales. En cualquier caso, la estructura más simple de matriz numérica es el vector, es decir, la matriz unidimensional.

En base a los conocimientos adquiridos en el capítulo anterior, proponemos ahora el desarrollo de un programa que, a partir de una serie de un número variable de elementos, gestione la extracción al azar de la totalidad de ellos, sin repetición. En el programa de ejemplo, hemos definido la variable **N** como 90, para simular las extracciones de bolas en el juego del bingo:



Organigrama de la rutina turno de jugadores.

Organigrama de la rutina de extracción aleatoria.




```
10 REM EXTRACCION ALEATORIA-
J.M.LOPEZ MARTINEZ
20 LET N=90: DIM S(N)
30 FOR I=1 TO N: LET S(I)=I: NEXT I
40 FOR I=1 TO N
50 LET X=INT (RND*(N-I))+1
60 PRINT ("      "+STR$ S(X))(LEN STR$
S(X) TO  )
70 FOR J=X TO N-I: LET S(J)=S(J+1):
NEXT J
80 NEXT I
```

En la línea 20, se define el número de elementos de la serie de aleatorios a producir, en nuestro caso 90. A continuación, dimensionamos un vector numérico **S( )**, por medio de la sentencia **DIM**, con un número **N** de elementos. Esta lista contendrá la serie ordenada desde el 1 hasta el valor que asignemos de comienzo a la variable **N**.
En la instrucción 30, se efectúa la carga de la matriz **S( )**, siendo el contenido de cada elemento idéntico al del índice que la califica, es decir, coincide el valor del elemento en cuestión con su posición relativa dentro del vector. Así por ejemplo, el elemento uno contiene el valor uno, el dos el valor dos, etc...
En las líneas 10 a 80 se encuentra el bucle central del programa, establecido para los valores de **I** comprendidos entre **1** y **N**, y en la línea 50, se efectúa la extracción aleatoria de un elemento de la serie, a través de la función **RND**, calculado con la fórmula siguiente: **LET X=INT(RND *(N-I))+1.**
Antes de comentar en detalle el por qué del empleo de esta fórmula, debemos hablar de las posibilidades que se nos ofrecen a la hora de extraer, de una serie de elementos, algunos de ellos, sin que éstos puedan repetirse.
La primera solución es la propuesta en el programa de ejemplo del capítulo anterior, donde a cada nuevo elemento seleccionado se le hace una marca en una posición determinada para, posteriormente, ejecutar una generación aleatoria controlada. Este control consiste en repetir una y otra vez la generación aleatoria, hasta que arroje como resultado un elemento de la matriz sin marca, es decir, inédito hasta el momento.
Lógicamente, a ninguno se nos escapa que este método es únicamente eficaz cuando, sobre una serie grande, se han de extraer unos pocos elementos, como es el caso de las diez preguntas sobre el total de los países europeos; esto es debido a que cada vez se genera un número aleatorio, comprendido en el intervalo que va desde uno al total de países europeos. Si en nuestro ejemplo actual empleamos el mismo procedimiento, según se van extrayendo los números, se hace más y más difícil, obtener uno que no haya salido ya.
Con el empleo de la fórmula de la línea 50, el margen se reduce progresivamente desde **N** has-

¡!

Una matriz de dos dimensiones, es un conjunto de conjuntos, es decir, una estructura formada por varios conjuntos de un nivel superior, representados por el primer índice, y varios de nivel inferior, representados por el segundo.

*

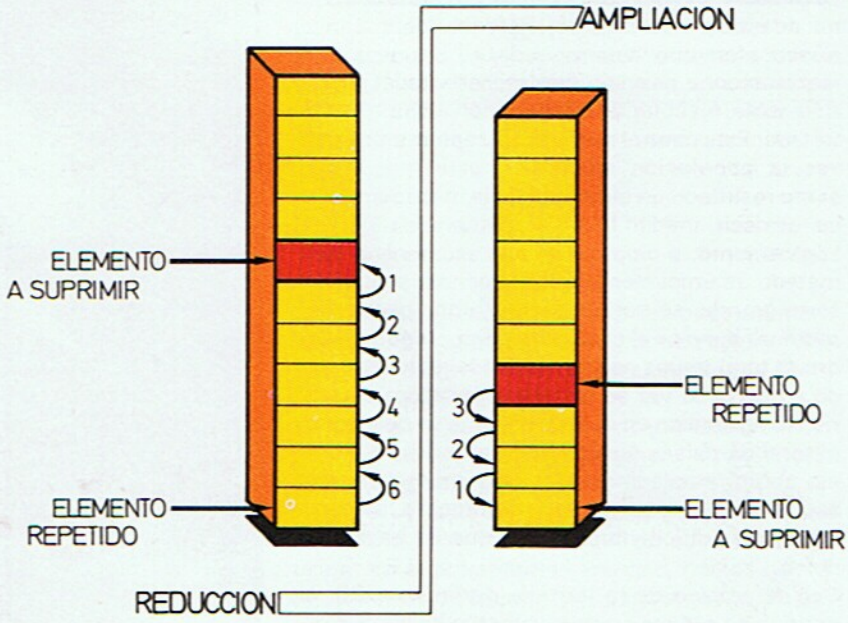
Al hablar de matrices de tres dimensiones, nos movemos en un conjunto que abarca a un subconjunto, y éste a su vez a otro subconjunto de él, de forma anidada y perfectamente estructurada.

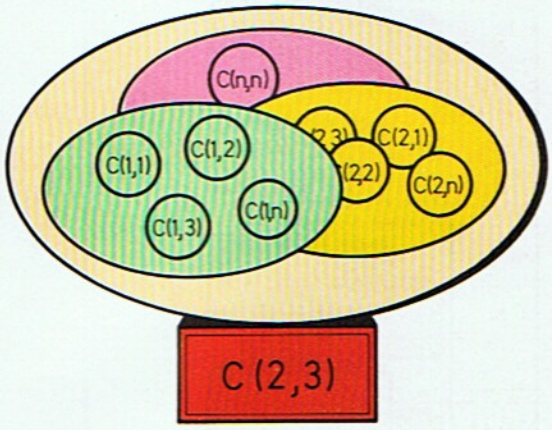
ta uno, por cada nueva pasada del bucle principal de programa **FOR NEXT**, obteniéndose mejores resultados. Sin embargo, supongamos que el primer número seleccionado no sea el último de la serie (el noventa en nuestro caso), cosa por otra parte harto probable; al reducir en uno el margen de operación de **RND** (de 1 a 89), descartaría la elección del 90, con lo cual este sistema es auténticamente catastrófico, puesto que sólo funcionará cuando el número elegido en cada ocasión sea el último del intervalo (90, 89, 88, 87, etc...). La explicación de por qué elegir este método, aparentemente absurdo, se encuentra en la línea 70, estudiada un poco más adelante, la cual sirve de complemento a la línea de extracción aleatoria, y hace que el nuevo sistema sea completamente válido.

En la línea 60 se imprime el elemento de la serie obtenido en la línea anterior, justificado por la derecha en cuatro posiciones, cosa que hacemos para que los números queden columnados y aparezcan 8 en cada línea. El algoritmo empleado para conseguirlo es: **PRINT (" "+STR$ S(X))(LEN STR$ S(X) TO).**

Generalizando, para todas las ocasiones en que nos interese obtener columnas de números ajustados a la derecha, podemos decir que el sistema consiste en montar una cadena, suma de tantos espacios como deseemos sea el ancho de la columna menos 1, más la **STR$** de la variable a imprimir, para escribir de toda ella la parte comprendida desde la **LEN** de la **STR$** al final de la cadena. ¿Complicado? Menos de lo que parece, sólo tenemos que practicar un poco con el algoritmo, para llegar a comprender y manejarlo a la perfección.

Según el desplazamiento de elementos de una matriz tienda a reducirla o a ampliarla, el **STEP** *del* **FOR** *empleado en el algoritmo, deberá ser negativo o positivo, respectivamente.*





La estructura matricial puede ser entendida matemáticamente como un conjunto de subconjuntos.

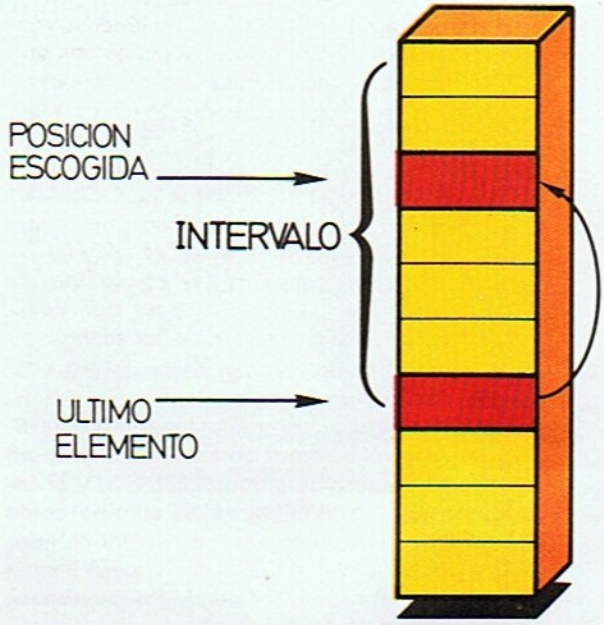
Como ya hemos dicho antes, en la línea 70 incluimos la rutina que asegura la validez del sistema de extracción aleatoria. Esta rutina, en esencia, elimina el elemento elegido de la serie, desplazando todos los demás que se encuentran detrás de él a la posición relativa anterior. De esta forma, la serie es cada vez más corta, manteniéndose siempre compuesta únicamente por elementos que todavía no han salido.

Si, por ejemplo, el primer número extraído es el tres, los elementos del 4 al 90 (fin del primer intervalo), pasarán a ocupar la posición anterior: el cuatro la tres, el cinco la cuatro, hasta el noventa la ochenta y nueve. Al mismo tiempo, esta línea nos ha servido para ilustrar la técnica de desplazamiento en elementos de una tabla, utilizada muy frecuentemente; tengamos en cuenta en este sentido, que el algoritmo empleado debe ser diferente según el intervalo se reduzca (programa de ejemplo) o se amplíe.

En este último caso, sólo varía el orden en que se alteran los elementos, que en vez de ascendente debe ser descendente. Supongamos que queremos desplazar las líneas de una matriz **S( )**, desde la línea 3 a la 89, para que ocupen las posiciones de la 4 a la 90; la instrucción adecuada sería: **FOR I= 90 TO 4 STEP -1: LET S(I)=S(I-1): NEXT I.**

Si no hubiera sido por querer otorgar un valor más pedagógico al ejemplo, podríamos haber utilizado un sistema bastante más sencillo para dar validez a la generación aleatoria: simplemente, sustituir el elemento extraído por el último del intervalo. Así, siguiendo el ejemplo anterior, si el número extraído hubiera sido el tres, y el último

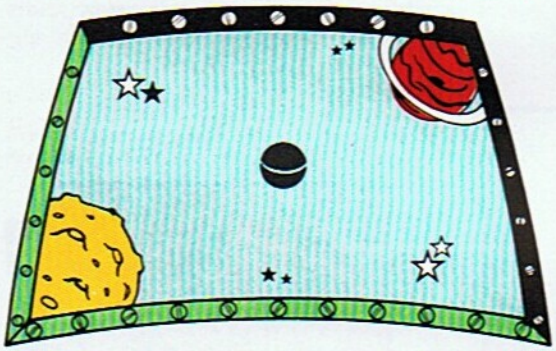
del intervalo el noventa, el problema se hubiera limitado a efectuar **LET S(3)=S(90)**, y generalizando, la línea 70 del programa, pasaría a ser **70 LET S(X)=S(N-I+1)**.
Por último, en la línea 80, se cierra el bucle principal del programa con la instrucción **NEXT I**. De esta forma, el bucle de la línea 70 queda anidado dentro del bucle exterior.
Como podemos ver, el hecho de haber definido el número de elementos de la serie mediante una variable **(N)**, nos permite alterarla fácilmente, flexibilizando así el programa, puesto que funcionará tanto con 90 elementos (ejemplo), como sería el caso de la extracción de bolas de bingo, como con 40, para la extracción de cartas de la baraja española, o cualquier otra aplicación similar.



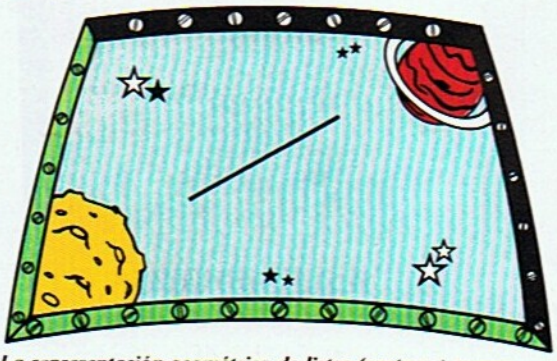
El sistema más fácil para dar validez al algoritmo de extracción aleatoria, habría sido hacer una copia en la posición escogida del último elemento del intervalo.

MATRICES NUMERICAS DE DOS DIMENSIONES

Para tener una idea clara de la estructura de una matriz de dos dimensiones, debemos pensar en que nos movemos dentro de un conjunto de conjuntos, es decir, tenemos varios conjuntos de un nivel superior, representados por el primer índice, y varios de nivel inferior, representados por el segundo.



Las variables, o la matrices unidimensionales de un solo elemento, pueden representarse geométricamente por un punto.



La representación geométrica de listas (vectores) es una línea.

Dado que para acceder a un elemento determinado debemos especificar sus dos índices, vemos claramente una analogía con la representación cartesiana en el plano, a través de la abscisa y la ordenada; debido a ello, este tipo de variables suscritas se conocen como TABLAS, puesto que gráficamente pueden ser representadas como una tabla formada por filas y columnas, o si los preferimos, como una serie de vectores situados uno al lado de otro.
A modo de ejemplo, proponemos un programa por el que se obtiene un informe sobre las calificaciones y notas medias de los 9 alumnos de una clase.

```
  10 REM ESTADISTICA DE CALIFICACIONES -
 J.M LOPEZ MARTINEZ
  20 DIM C(9,8)
  30 PRINT TAB 7;"- CARGA DE DATOS -"
  40 FOR L=0 TO 31: PRINT "=";: NEXT L
  50 PRINT ''
  60 FOR J=1 TO 9: FOR K=1 TO 8
  70 INPUT "Alumno "+STR$ J+" NOTA "+STR
$ K+":";C(J,K): IF C(J,K)<0 OR C(J,K)>10
 THEN  GO TO 70
  80 PRINT ("    "+STR$ C(J,K))(LEN STR$
C(J,K) TO );
  90 NEXT K: PRINT '': NEXT J
 100 CLS : PRINT "NUM";
```

¡!

Los vectores o matrices unidimensionales, son la expresión más simple de una estructura de conjunto. Consisten en una serie ordenada de un número determinado de elementos, calificados por un índice.

*

La única limitación dimensional en las matrices la impone la propia memoria; por tanto, podemos crear estructuras de datos de extraordinaria complejidad.

¡!

En general, cualquier estructura matricial, se puede entender como un árbol invertido, en el que cada índice genera las ramas correspondientes a otros índices.

*

Las estructuras matriciales de dos dimensiones, son representadas usualmente como tablas, en las cuales uno de los índices son las filas y el otro las columnas, por tanto, forman geométricamente un plano.

*

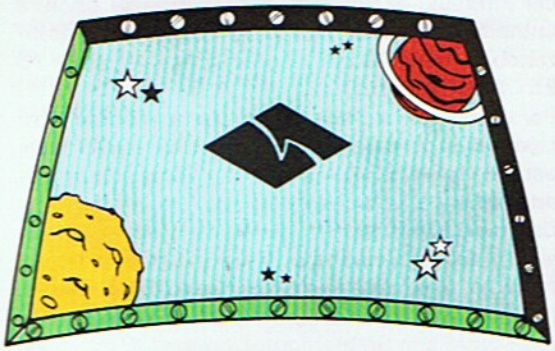
También es usual representar las estructuras matriciales como conjuntos de subconjuntos.

```
 110 FOR K=1 TO 8: PRINT (' N'+STR$ K)(L
EN STR$ K TO );: NEXT K
 120 PRINT ' N.M'
 130 FOR L=0 TO 31: PRINT '=';: NEXT L
 140 FOR J=1 TO 9
 150 LET S=0: PRINT ('  '+STR$ J)(LEN ST
R$ J TO );
 160 FOR K=1 TO 8
 170 LET S=S+C(J,K): PRINT ('  '+STR$ C(
J,K))(LEN STR$ C(J,K) TO );
 180 NEXT K
 190 LET X$=STR$ INT ((S/8+.05)*10): LET
 X$=('   '+X$)(LEN X$ TO ): PRINT X$( TO
 3);'.';X$(4)''
 200 NEXT J
```

En la línea 20 se define, por medio de una instrucción **DIM**, la matriz numérica de dos dimensiones **C(9,8)**. Al corresponder un índice por dimensión, nos referiremos en adelante al primero de ellos (el número de alumno) como **J**, y al segundo (el número de nota de cada alumno) como **K**. La matriz tiene, pues, una estructura de nueve filas por ocho columnas, accediéndose a cada elemento de la misma, indicando los correspondientes valores de **J** y **K** (alumno y nota).

En las líneas 30 a 50 se gestiona la cabecera de la carga de datos. Primero se imprime un título centrado y, seguidamente, una línea de guiones tras la cual se produce una de separación.

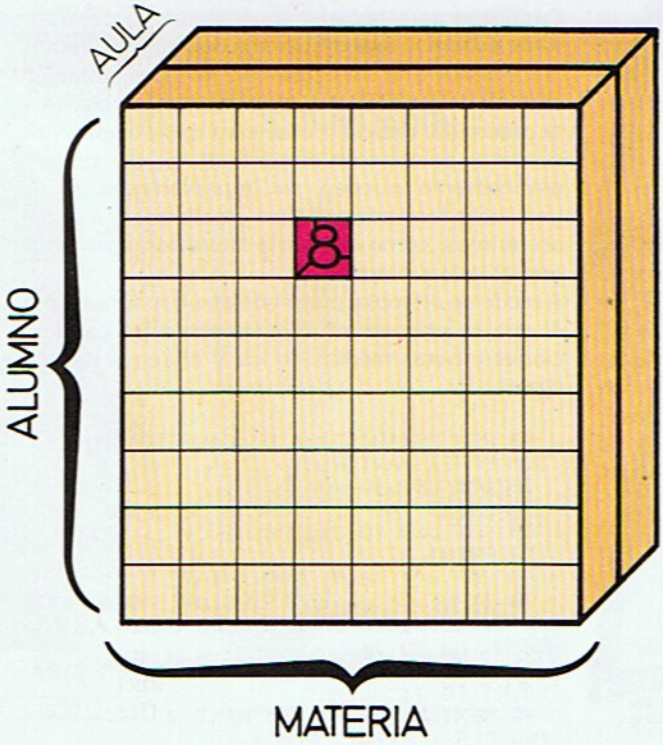
En la líneas 60 a 90 se gestiona la entrada de datos propiamente dicha. La 60 abre los bucles **FOR NEXT** anidados de las variables **J** y **K**, los cuales realizan la carga ordenada de los datos en la tabla, gracias a la siguiente instrucción, que contiene el **INPUT**.



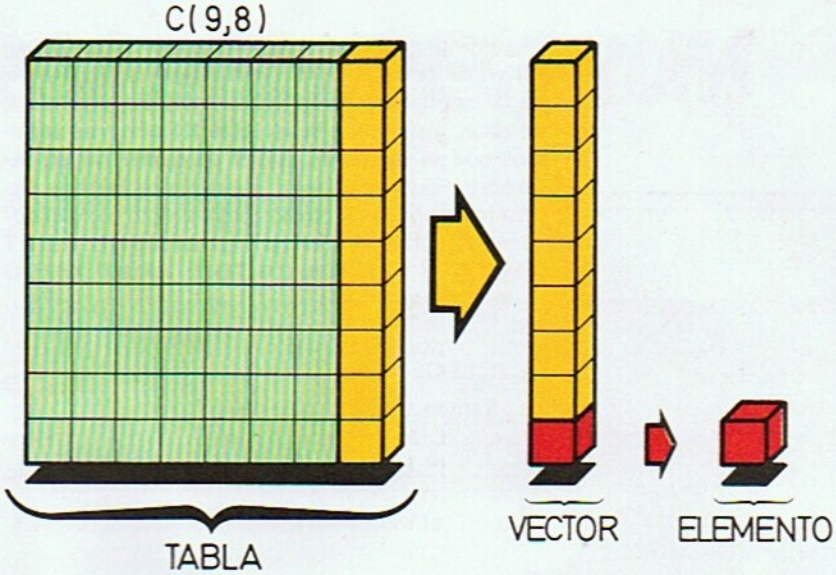
La representación geométrica escogida para una matriz bidimensional, es un plano.

La instrucción 70 se divide en dos partes bien diferentes. En la primera de ellas, se aceptan los datos con un mensaje de petición, construido a base de las palabras descriptivas de los índices y los respectivos valores de éstos mismos, conociéndose de este modo, en todo momento, el dato exacto a teclear. En la segunda parte, se efectúa la depuración mínima necesaria para obtener una cierta garantía sobre la validez de los datos introducidos. En este caso, como el **INPUT** es de tipo numérico, no es necesario verificar que los datos introducidos lo sean. De esta forma, con-

El aula es el tercer índice utilizando en el programa de ejemplo para determinar un elemento de la matriz.



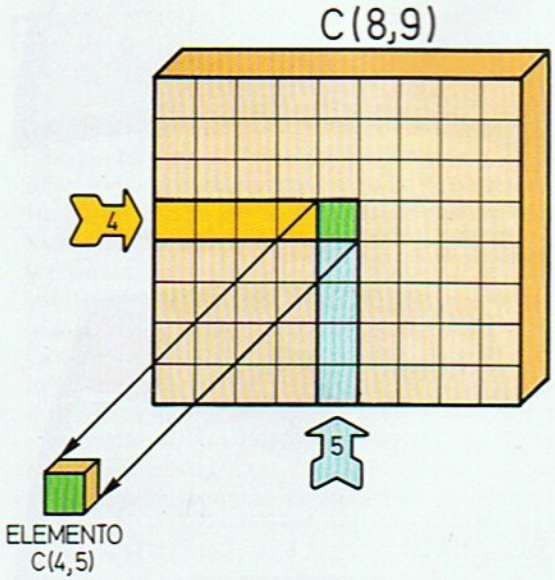
centramos nuestros esfuerzos en controlar que éstos se encuentren dentro de los márgenes razonables (0 a 10).
En la línea 80 se van imprimiendo los datos introducidos justificados a la derecha, dentro de columnas de 3 posiciones cada una, completando así una línea por cada 8 calificaciones introducidas.
La línea 90 cierra los bucles anidados de notas y alumnos, controlando un espacio doble entre líneas por medio de la instrucción **PRINT''**, concluyendo en esta línea la carga de datos en la matriz.
Las líneas de la 100 a la 200 gestionan la impresión de la estadística de resultados obtenidos. Dentro de ellas, las líneas 100 a 130 imprimen una cabecera para los resultados. Esta cabecera se construye, partiendo del borrado de la pantalla y la impresión de un título, para la columna destinada a albergar los números de los alumnos. A continuación, se imprimen una serie de ocho cabeceras formadas por la inicial **N** seguida del número de la columna. Por último, recurrimos a la misma rutina de subrayado con el símbolo igual (=), ya empleada en ocasiones anteriores.
En las líneas 140 a 200 se desarrolla verdaderamente la actividad de impresión de calificaciones y cálculo e impresión de la nota media: la 140 comienza con el bucle exterior de alumnos; seguidamente, la instrucción 150 imprime el comienzo de la línea de detalle de calificaciones de cada alumno, consistente en su número centrado frente a la cabecera, haciendo cero la variable **S**, que contiene la suma de las notas de cada alumno para el cálculo de la media.



La matriz numérica puede utilizarse como conjunto de varios vectores (tabla). Tal es el caso de la matriz C(9,8) en el programa de ejemplo.

En las líneas 160 a 180 codificamos el bucle de impresión de calificaciones columnadas por la derecha, con acumulación en la variable **S** de cada nota.
Por último, en la instrucción 190 resolvemos la impresión de la nota media, después de efectuar su cálculo. La operación es simple, bastando con dividir **S** por ocho. Sin embargo, nosotros la hemos complicado un poco, al asegurarnos de que las notas medias se impriman redondeadas y con un decimal. Para poder hacerlo, utilizaremos el algoritmo: **INT((S/8+.05)*10)**.
Este algoritmo puede utilizarse de forma general para imprimir los valores numéricos redondeados a un número fijo de decimales. Para ello, debemos tener en cuenta que a la variable en cuestión es preciso sumarle un 5 en la posición decimal siguiente por la derecha a la que pretendemos redondear el resultado. En nuestro ejemplo, sumamos .05 porque queremos redondear a un decimal. Si quisiéramos redondear a enteros escribiríamos .5 y para redondear el resultado a dos decimales .005.
Una vez hecho esto, basta con multiplicar el resultado obtenido por 10, 100,... para obtener uno, dos... decimales, y por último, calcular la parte entera de lo obtenido hasta el momento. Con esto concluye la fase de cálculo, pero aún falta imprimir los resultados, es decir, interpretar el contenido de la variable alfanumérica en la cual hemos colocado el resultado de las operaciones, en nuestro caso **X$**, cuyo valor es la **STR$** del **INT** del valor numérico obtenido.

Para designar cualquier elemento de una tabla, son precisos dos índices.



Para acceder a un elemento determinado debemos especificar los dos índices, una clara analogía con la representación cartesiana en el plano, a través de la abscisa y la ordenada.

Las matrices tridimensionales precisan de tres índices para calificar a cada elemento, lo que implica cierta relación con la representación espacial pura, en tres dimensiones.

Ya ciñéndonos a nuestro ejemplo, en la segunda mitad de la línea 190, efectuamos la impresión de la media con dos instrucciones. En la primera de ellas, ajustamos la variable **X$** con tres espacios por su izquierda, antes de comenzar su interpretación; hecho esto, imprimimos la parte entera (tres primeras posiciones), un punto, y el decimal (la cuarta posición): **LET X$=(" "+X$)(LEN X$ TO): PRINT X$(TO 3);".";X$(4)''.**

MATRICES NUMERICAS DE TRES DIMENSIONES

Las matrices tridimensionales precisan de tres índices para calificar a cada elemento. Si hemos dicho que las bidimensionales, tienen cierta similitud con la representación cartesiana en el plano, podemos decir ahora que éstas la tienen con la representación espacial pura, en tres dimensiones. Nos movemos, en este caso, en un conjunto que abarca a un subconjunto, y éste a su vez a otro subconjunto de él, de forma anidada y perfectamente estructurada.

Volviendo al ejemplo anterior de las calificaciones, podemos proponer ahora una ampliación al problema planteado anteriormente, donde el análisis se extiende a las dos aulas que componen el curso. Definimos de esta forma una matriz tridimensional **C(2,9,8)**, e índices **I, J** y **K.**

Por lo demás, el desarrollo del programa es bastante similar al anterior, con la inclusión de una rutina de borrado parcial de la pantalla, para admitir separadamente las calificaciones de las dos aulas, tal como podemos observar en el correspondiente listado.

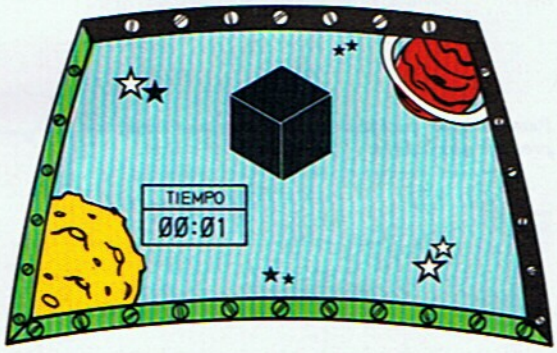
Las estructuras matriciales unidimensionales (vectores o listas), se pueden representar como una línea, o como un punto en el caso particular de un vector con un solo elemento.

Las estructuras matriciales tridimensionales, se pueden representar gráficamente como un volumen, en el cual uno de los índices corresponde a la altura, otro a la anchura y el restante a la profundidad. Así pues, representan una figura en el espacio y ya no en el plano.

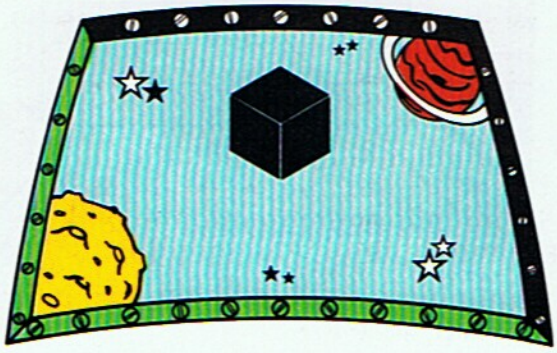
```
 10 REM ESTADISTICA DE CALIFICACIONES -
J.M.LOPEZ MARTINEZ
 20 DIM C(2,9,8): LET S$="": FOR L=0 TO
31: LET S$=S$+" ": NEXT L
 30 PRINT TAB 7;"- CARGA DE DATOS -"
 40 FOR L=0 TO 31: PRINT "=";: NEXT L
 50 PRINT ''
 60 FOR I=1 TO 2: PRINT AT 2,0: FOR L=0
 TO 16: PRINT S$: NEXT L: PRINT AT 2,0:
FOR J=1 TO 9: FOR K=1 TO 8
 70 INPUT "Curso "+STR$ I+" alumno "+ST
R$ J+" NOTA "+STR$ K+":";C(I,J,K): IF C(
I,J,K)<0 OR C(I,J,K)>10 THEN  GO TO 70
 80 PRINT ("    "+STR$ C(I,J,K))(LEN STR
$ C(I,J,K) TO );
 90 NEXT K: PRINT '': NEXT J: PAUSE 0:
NEXT I
100 CLS : PRINT "NUM";
110 FOR K=1 TO 8: PRINT (" N"+STR$ K)(L
EN STR$ K TO );: NEXT K
120 PRINT " N.M"
130 FOR L=0 TO 31: PRINT "=";: NEXT L
140 FOR I=1 TO 2: PRINT AT 2,0: FOR L=0
 TO 16: PRINT S$: NEXT L: PRINT AT 2,0:
FOR J=1 TO 9
150 LET S=0: PRINT ("  "+STR$ J)(LEN ST
R$ J TO );
160 FOR K=1 TO 8
170 LET S=S+C(I,J,K): PRINT ("  "+STR$
C(I,J,K))(LEN STR$ C(I,J,K) TO );
180 NEXT K
190 LET X$=STR$ INT ((S/8+.05)*10): LET
 X$=("    "+X$)(LEN X$ TO ): PRINT X$( TO
 3);".";X$(4)''
200 NEXT J: PAUSE 0: NEXT I
```

Para finalizar añadiremos que las matrices no presentan una restricción a tres dimensiones, sino que también podemos construir estructuras tetradimensionales, etc... Así por ejemplo, podríamos conservar en una misma matriz las notas obtenidas por todos los alumnos en todas las asignaturas para las diferentes clases, y además, durante todo el curso; en este caso, la cuarta dimensión escogida sería el tiempo (los meses del curso).

Las representaciones tetradimensionales son extraordinariamente más complejas que las vistas hasta ahora; un caso habitual es considerar el tiempo como la cuarta dimensión.



Las matrices tridimensionales son representadas geométricamente con volumen.



LAS IMPRESORAS DEL SPECTRUM

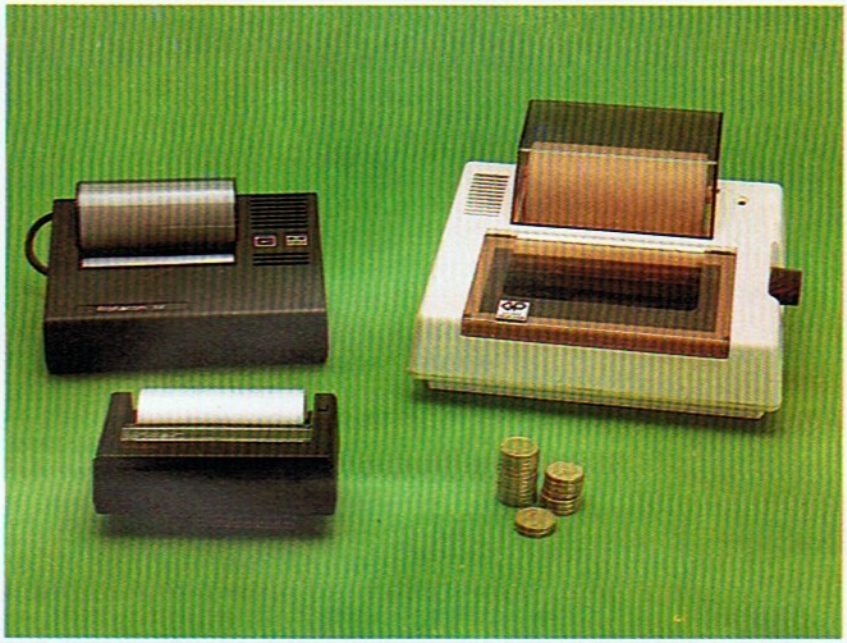
CON el paso del tiempo, entre los usuarios del Spectrum circula una idea generalizada: «A mi ordenador le falta algo». Debido a esto, tanto Sinclair, fabricante de nuestro micro, como otras firmas dedicadas a la producción de hardware, lanzan constantemente al mercado nuevos periféricos, con los que satisfacer la creciente demanda de los usuarios; eso sí, a costa de aumentar la confusión por parte del comprador, dado el elevado número de complementos disponibles. En las páginas de esta sección, tratamos de orientar a todos aquellos que en determinado momento deban tomar una decisión sobre este o aquel equipo accesorio a nuestro Spectrum, así como a los que puedan ya poseerlo, con el fin de esclarecer sus posibilidades de manejo, yendo más allá de las escuetas explicaciones que adjuntan los manuales de instrucciones que, por si además fuera poco, en muchos casos se encuentran en inglés.

Cuando tratamos el tema de los *joysticks*, vimos como, inevitablemente, fuera cual fuera el modelo elegido, debíamos disponer de un interface. Lo mismo ocurría con los discos; incluso en el capítulo dedicado como introducción a las impresoras, hacíamos hincapié en los interfaces más utilizados por estas.

Sin duda es la impresora el periférico más ampliamente difundido, entre los ordenadores a los que se les pretende encomendar una misión más allá del genocidio extraterrestre. Debido a ello, tres firmas comerciales han decidido lanzar al mercado unas impresoras que, sin necesidad de interface especial, puedan ajustarse a las necesidades básicas del usuario del Spectrum. Todas ellas con una característica común: su bajo coste. No quiere decir esto que no dispongan de interface. Este va acoplado dentro del propio equipo, y además, está diseñado para funcionar exclusivamente con nuestro Spectrum, gracias a lo cual, el precio del periférico disminuye sensiblemente. Las tres impresoras mencionadas, disponen del juego de caracteres completo de nuestro ordenador, incluyendo mayúsculas, minúsculas, caracteres gráficos, en vídeo inverso, etc. Por tanto, admiten la posibilidad de conseguir caracteres subrayados o acentuados, de la misma manera que lo haríamos para presentarlos en la pantalla del televisor o monitor.

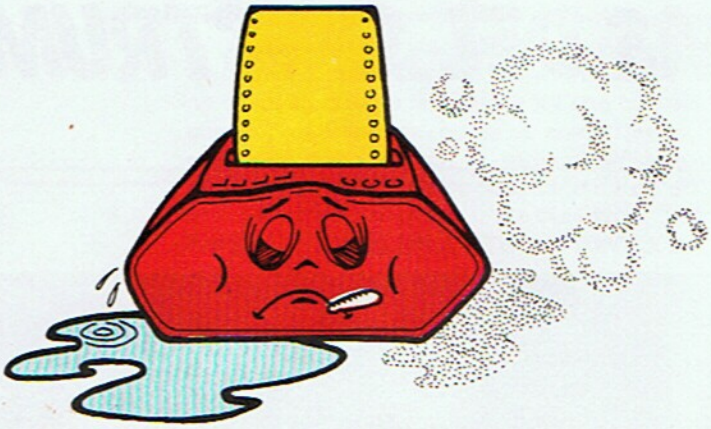
Para su control, se utilizan los comandos LLIST y LPRINT, equivalentes a LIST y PRINT, aunque referidos a la impresora y no a la pantalla. En este aspecto, es conveniente hacer una pequeña puntualización. La salida de información en el Spectrum, se efectúa a través de distintas corrientes; así, las corrientes 0 y 1 están asociadas a la parte inferior de la pantalla, la 2 a su parte superior, y la 3 a la impresora. Las restantes, no se encuentran asignadas a ningún canal (periférico) específico.

Según el periférico que deseemos emplear, el Spectrum se encarga de seleccionar la corriente adecuada. De esta manera, PRINT habitualmente se remite a la zona superior de la pantalla; po-



Las tres impresoras que tratamos en este capítulo, tienen en común su bajo precio.

Los periféricos (CANALES) se asocian al ordenador a través de CORRIENTES.



La humedad y el polvo son los grandes enemigos de nuestros periféricos.

demos modificar este estado de cosas, simplemente indicando la corriente concreta por la cual deseamos enviar la información, mediante **PRINT # <número de corriente>**. Por tanto, la desviación de datos a la impresora, puede conseguirse con **PRINT #3,** o lo que es lo mismo: **LPRINT** es una forma abreviada de escribir **PRINT #3.**

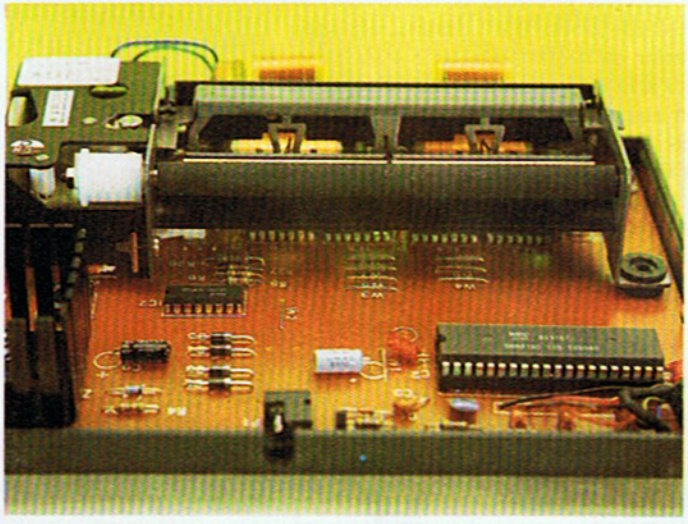
Si queremos compatibilizar un programa, de forma que escriba sus resultados en la impresora en vez de en la pantalla, bastará con ejecutar previamente la orden **OPEN #2, "p"**, gracias a la cual, el flujo que habitualmente se envía a la pantalla (corriente 2 asociada al canal "s"), se bifurca a la impresora, tal como indica el parámetro **"p"** ("p" como abreviatura de *printer*) a continuación de **OPEN**. Volvamos ahora al tema fundamental del capítulo.

Todas las impresoras que estamos tratando implementan el comando **COPY**, el cual efectúa una copia en papel del contenido de la pantalla, sin tener en cuenta, lógicamente, el color de la misma, sino sólo el estado del archivo de imagen, no incluyendo las dos últimas filas, reservadas por lo general al Sistema. Por otra parte, al ser periféricos especialmente diseñados para el Spectrum, todos ellos siguen el formato de impresión en treinta y dos columnas por fila.

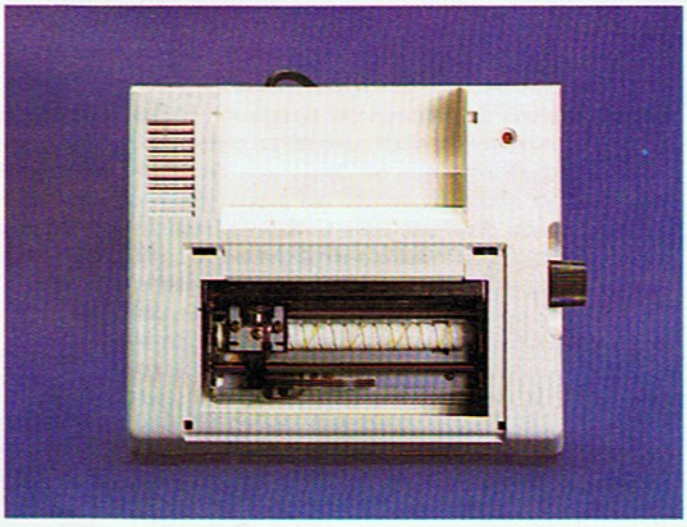
UNA CHISPA IMPRESIONANTE

Bueno, lo cierto es que no se trata de una enorme descarga eléctrica, que pueda hacer peligrar nuestra integridad física en el momento de poner en funcionamiento la impresora. Sencillamente, es el sistema empleado por la ZX PRINTER (impresora ZX) en la composición de los caracteres. Se trata de una impresora electrostática, es decir, cada punto de los que conforman un carácter, se genera al incidir sobre el papel una diminuta chispa eléctrica, poducida entre una placa fija, cargada, y una pequeña aguja que se desliza a lo ancho del papel.

El papel utilizado es especial, recubierto en una de sus caras por una fina capa de compuestos metálicos, de forma que al recibir la descarga, «revela» un punto, cambiando a una tonalidad más oscura. Su anchura máxima es de 100 mm. En el caso de la ZX PRINTER, el elemento impresor no es en realidad una sola aguja, sino dos, que realizan idéntico cometido. Estas se sitúan en oposición una de la otra, sobre una pequeña

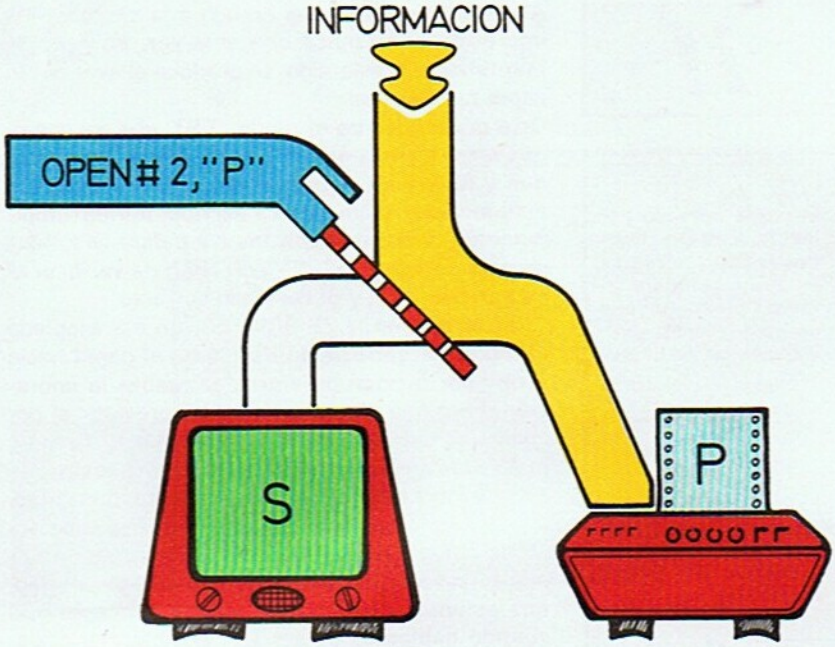


Debido a su sistema de impresión, la Alphacom 32 pertenece al tipo denominado «de línea».



Como se puede observar en la foto, la impresora Seikosha GP-50 es del tipo matricial.

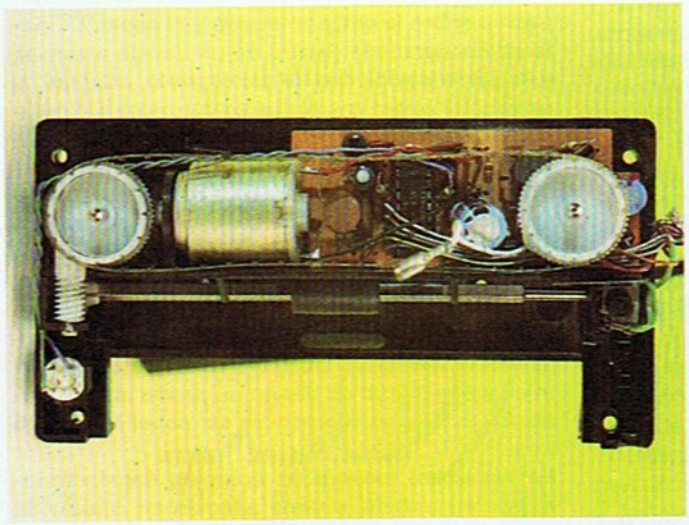
correa dentada de material plástico, apoyada sobre dos engranajes, uno de los cuales va unido al motor que suministra a la correa una velocidad de giro constante, desplazando de esta manera las agujas horizontalmente sobre el papel. Al mismo tiempo, cada vez que una de las agujas completa una pasada, un eje, igualmente engranado al motor, se encarga de desplazar hacia arriba el papel, con el fin de posicionar la siguiente línea de impresión. El sistema resulta, sin duda, ingenioso, y dada la simplicidad del mismo, los resultados, mejores de lo que cabría esperar, siempre y cuando nuestro objetivo no sea otro que el de obtener cortos listados, o sacar algún que otro comentario por impresora.
Moviéndonos en el campo de los pequeños detalles, diremos que el nivel del ruido es bajo, y en ningún caso se hace molesto. La calidad de la impresión la hace cuando menos legible, aunque se observan algunas deficiencias a la hora de trazar líneas verticales. Hoy en día, esta impresora que fue una gran ayuda, en su momento, para los pioneros del Spectrum, está prácticamente en desuso, dejando paso a otros equipos que la superan en velocidad y calidad.



El flujo de información que habitualmente discurre por PRINT *hacia la pantalla (canal "s"), puede ser fácilmente desviado a la impresora con* OPEN #)2,"p" *(canal "p").*

CARACTERES CALIENTES

Otra posible solución económica a la hora de adquirir una impresora, la proporciona la ALPHACOM 32. De nuevo, se trata de un aparato compacto, de fácil manejo, totalmente compatible con nuestro Spectrum. Todo el equipo va encerrado dentro de una carcasa de plástico inyectado, especialmente resistente a los golpes. De él parte el conector acoplable a la tarjeta de expansión de nuestro micro. Para su funcionamiento, es necesaria una fuente de alimentación externa, suministrada junto con la impresora.
Posee un pulsador OFF, cuya misión es desactivar el paso de información hacia la impresora, aunque esto no, quiere decir que quede desconectada de la alimentación. El botón ON/



El mecanismo de impresión de la ZX Printer, se basa en dos agujas montadas en posición opuesta sobre una polea.

Cada una de las impresoras estudiadas emplea un tipo de papel diferente: normal (Seikosha GP-50), térmico (Alphacom 32) y electrostático (ZX Printer).

¡!

El papel utilizado por las impresoras electrostáticas y térmicas es muy sensible al tacto, provocándose por esta causa deficiencias en la impresión.

*

Es importante seleccionar un papel de calidad adecuada para conseguir una buena impresión.

*

En las impresoras Seikosha GP-50S y Alphacom 32, debido a la distinta separación entre puntos horizontales y verticales, las imágenes reproducidas mediante **COPY**, se distorsionan ligeramente, alargándose en sentido vertical.

*

Las impresoras, como cualquier periférico, no deben ser conectadas al bus de expansión del Spectrum sin haber cortado previamente la alimentación a ambos aparatos.

ADVANCE, retornará el equipo a la situación de impresión si se pulsa una sola vez; en caso de mantenerse presionado, se produce el avance del papel hacia arriba.

Otra posibilidad es el modo TEST, que se consigue presionando, simultáneamente, ambos botones y liberando, a continuación, el de OFF. De este manera, la impresora escribe, ininterrumpidamente, hasta que volvamos a pulsar OFF, filas de caracteres "8" y "1", con el fin de verificar el estado del papel y el centrado correcto.

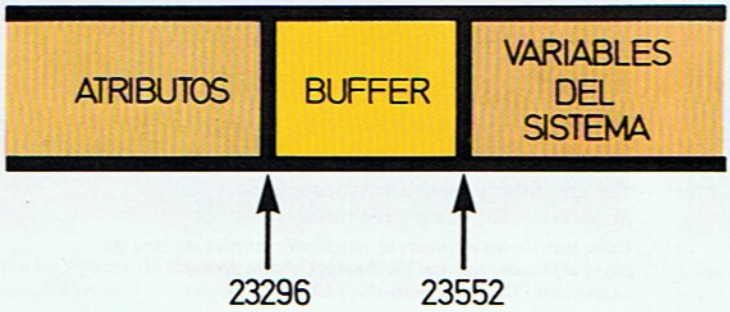
Al igual que en la ZX PRINTER, un eje acoplado al motor se encarga de desplazar el papel hacia arriba por fricción, mientras se realiza la impresión. Pero hay algo que no se ha previsto: si por cualquier causa queremos rebobinar el rollo de papel, la operación será imposible, puesto que este eje no permite el giro en sentido inverso.

Para realizar la impresión, utiliza una técnica denominada de líneas, es decir, se imprime una línea cada vez. Conviene aclarar que al decir línea, nos estamos refiriendo al mismo concepto que cuando hablamos de las 175 posibles en el esquema de alta resolución del archivo de imagen. De esta manera, la velocidad de impresión aumenta considerablemente, y la ALPHACOM 32 es capaz de escribir 16 líneas enteras (dos filas de caracteres completas) cada segundo, equivalentes aproximadamente a 64 caracteres por segundo. Frente a esta rapidez considerable, el inconveniente estriba en que empleará el mismo tiempo en imprimir, por ejemplo, una sola "A" que toda una fila de éstas. No obstante, la velocidad de impresión la sitúa como la más rápida dentro de su categoría.

El papel utilizado, nuevamente, es especial. En este caso se trata de papel térmico, es decir, recubierto por una de sus caras de sustancias termosensibles, que cambian de color cuando reciben una determinada cantidad de calor. Los resultados, en lo concerniente a calidad gráfica, y teniendo en cuenta el tamaño de la impresora, pueden considerarse excelentes. La anchura máxima del papel utilizado, no debe superar los 109 mm.

La tecnología térmica supone otra de las virtudes del equipo, puesto que, a la vez de tener menos elementos mecánicos móviles, ventaja contra las averías, el nivel sonoro del proceso impresión, disminuye a límites bajísimos.

La zona de memoria que el Spectrum emplea como intermediario entre la impresora y el ordenador, se denomina buffer de la impresora.



ECONOMICA DE IMPACTO

La SEIKOSHA GP-50S puede considerarse como la hermana pequeña, dentro de la amplia gama de impresoras matriciales, que esta firma japonesa ha comercializado en los últimos años.

Se trata de un equipo compacto, especialmente diseñado para el ZX 81 y el ZX Spectrum. También precisa una fuente de alimentación exterior, que suministra la energía necesaria para hacer funcionar el aparato. Un interruptor ON/OFF y un indicador luminoso, sirven para verificar que la impresora se ha conectado correctamente a la alimentación. En el caso de producirse algún error, la lámpara parpadeará mientras éste no sea subsanado.

Cuando se pulsa ON, la impresora realiza una secuencia de inicialización, desplazando la cabeza de impresión dos veces a lo ancho del papel, quedando de esta manera, preparada para la escritura.

El cabezal de impresión está formado por una matriz de 7×7 pequeñas agujas, más otra adicional utilizada en gráficos de alta resolución; éstas impactan sobre la cinta entintada y el papel. La cinta va encapsulada dentro de un casete especial, y existe la posibilidad de conseguirla, además de en el color negro con el que se suministra el equipo, en distintos colores tales como el rojo, naranja, azul, verde, etc.

Por tratarse de una impresora matricial, el nivel de ruido aumenta considerablemente respecto a las anteriores. La velocidad de impresión es de 35 caracteres por segundo, y la calidad gráfica, aceptable. Tiene la ventaja de admitir papel normal, cuestión a tener en cuenta a la hora de pensar en el recambio, pues los rollos de papel térmico y electrostático tienen un precio más elevado. En este caso, la anchura del papel es de 126 mm.

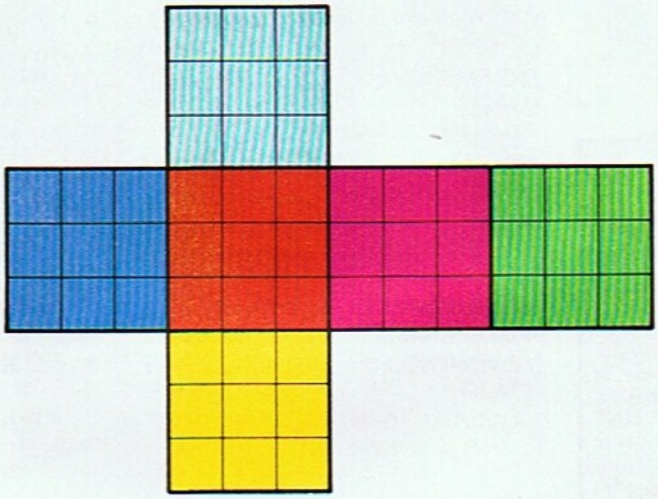
En resumen, todas ellas cumplen satisfactoriamente su trabajo, y tiene además en común un bajo coste, aunque lógicamente, no debemos esperar resultados excesivamente espectaculares.

EL CUBO MAGICO

SEGURAMENTE, en alguna ocasión hemos tenido en nuestras manos el famoso CUBO DE RUBIK. Hemos dado vueltas y más vueltas a cada uno de los bloques en que está dividido, y solamente en contadas ocasiones hemos logrado reordenar satisfactoriamente los nueve cuadritos que componen cada una de las seis caras. Con este nuevo programa, tendremos plena garantía de alejar el aburrimiento.

Aunque este juego ha hecho auténtico furor entre la juventud europea, no daremos por supuesto que las nuevas generaciones conocen completamente su funcionamiento, y brindaremos unas breves explicaciones sobre el objetivo de este pasatiempo. El protagonista del juego es un cubo móvil (seis caras), compuesto por nueve cuadritos de color por cada cara; todas las caras pueden girar sobre sí mismas, en el sentido horario o antihorario.

Cada una de las caras está compuesta por tanto por nueve bloquecitos de un mismo color (por supuesto diferente en cada una de ellas), que se pueden desplazar mediante un giro. El cubo inicia el juego con una zona de cada color, posteriormente se desordena de una manera aleatoria, con varios movimientos, y acto seguido, se puede pasar a ordenarlo nuevamente. Aquí llega el problema: desordenarlo es bien fácil, pero girar las caras adecuadas para que todos los colores vuelvan a quedar en su lugar, no es precisamente sencillo.



En el desarrollo del cubo que se presenta en la pantalla, la cara roja coincide con la superior.

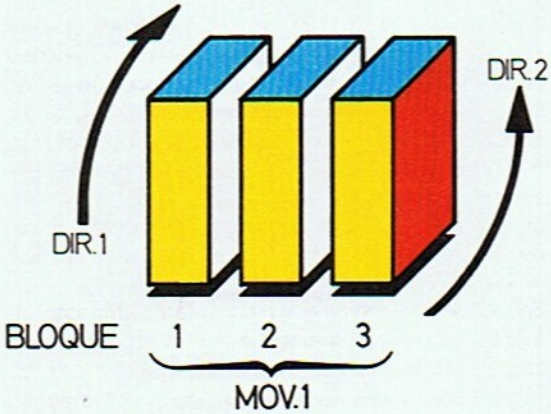
EL CUBO DEL SPECTRUM

La representación en la pantalla de una figura con volumen es evidentemente muy compleja, de ahí que hayamos escogido como solución, representar el cubo en desarrollo, es decir, en forma de cruz, tal como se muestra en la figura. Esto implica, que la cara roja representa la parte superior, la azul claro la posterior, la amarilla la anterior, la verde la inferior, la magenta la derecha y la azul oscura la izquierda.

Una vez conocida la situación del cubo, podemos comenzar el programa. Como primera medida se nos solicita la elección del nivel de dificultad, el cual consiste en el número de movimientos aleatorios a que se someterá el cubo en su fase de desorden. Una vez escogido el nivel, el cubo se presentará en la pantalla, y se desordenará, para a continuación pasar a la fase de resolución del problema por parte del jugador.

La codificación de nuestra jugada, es decir, la ma-

El primer movimiento se considera en anchura.



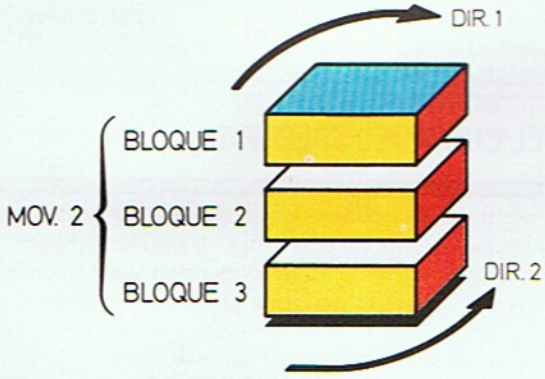
nera en que vamos a indicar al Spectrum el movimiento que queremos realizar, es bastante fácil de aprender, y pronto nos acostumbraremos a ella. Lo más práctico será que sigamos las indicaciones que se brindan a continuación, auxiliados por los gráficos explicativos.
En primer lugar hemos de indicar el tipo de movimiento a efectuar. Este dependerá del plano en que deseemos realizar la jugada:

MOVIMIENTO 1. GIRO EN PLANO Z-X (anchura)
MOVIMIENTO 2. GIRO EN PLANO X-Y (altura)
MOVIMIENTO 3. GIRO EN PLANO Y-Z (profundidad)

Cada uno de estos movimientos, podría señalar el desplazamiento de cualquiera de los tres bloques que se van a ver afectados. Así pues, el siguiente paso es indicar el bloque a desplazar (de 1 a 3):

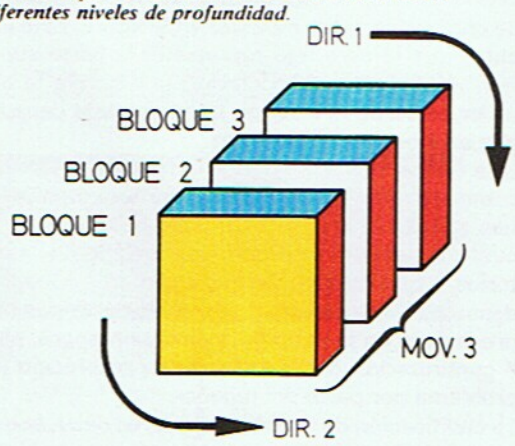
MOVIMIENTO 1. NUMERADOS DE IZQUIERDA A DERECHA
MOVIMIENTO 2. NUMERADOS DE ARRIBA A ABAJO
MOVIMIENTO 3. NUMERADOS DE PRIMER PLANO A FONDO

Finalmente, sólo queda especificar el sentido de giro del bloque escogido. Y en este caso, se cumple indudablemente que una imagen vale más que cien palabras, de manera que lo más sencillo será que observemos los gráficos adjuntos, donde se detalla mediante una flecha, qué sentido es el 1 y cuál el 2, según cada movimiento.
Por cada jugada que ejecutamos, el ordenador efectúa una considerable cantidad de operaciones: calcula el efecto del movimiento, actualiza las seis tablas que tiene definidas, controla si el cubo ha sido solucionado, incrementa el contador de jugadas y, finalmente, muestra el movimiento especificado.
Esta serie de tareas, añadido a la estructura del programa, en forma de subrutinas, hace que el desplazamiento del cubo no sea inmediato, pero sin duda lo suficientemente ágil para lo que se pretende, sobre todo teniendo en cuenta que en este juego se confirma un conocido refrán: «Más vale maña que fuerza». Efectivamente, en el juego original siempre nos cabía la posibilidad de desmontar el cubo y volver a montarlo ordenado, sin embargo ahora...
La introducción del programa, sólo tiene el problema de las letras subrayadas que aparecen en el listado; como ya sabemos, éstas corresponden a los caracteres gráficos de las teclas afectadas. A modo orientativo, señalaremos que dichos caracteres se encuentran en las siguientes líneas: 80, 90, 130, 140, 170, 180, 6875, 6880, 6885, 6890, 6895, 6900, 6905, 6910, 6945, 6950, 6955 y 6960. Para la grabación del programa, utilizaremos el comando **SAVE "CUBO MAG."** (sin autoejecución), o bien **SAVE "CUBO MAG." LINE 1** (con autoejecución a partir de la primera línea).



El movimiento dos se efectúa en las diferentes alturas.

El tercer tipo de movimiento se lleva a cabo en los diferentes niveles de profundidad.



¡!

Para grabar el programa, utilizaremos la instrucción **SAVE "CUBO MAG." LINE 1**. Si suprimimos la especificación final **LINE 1**, el programa no se autoejecutará al finalizar la carga, aunque podrá ser lanzado mediante **RUN**, sin ninguna dificultad.

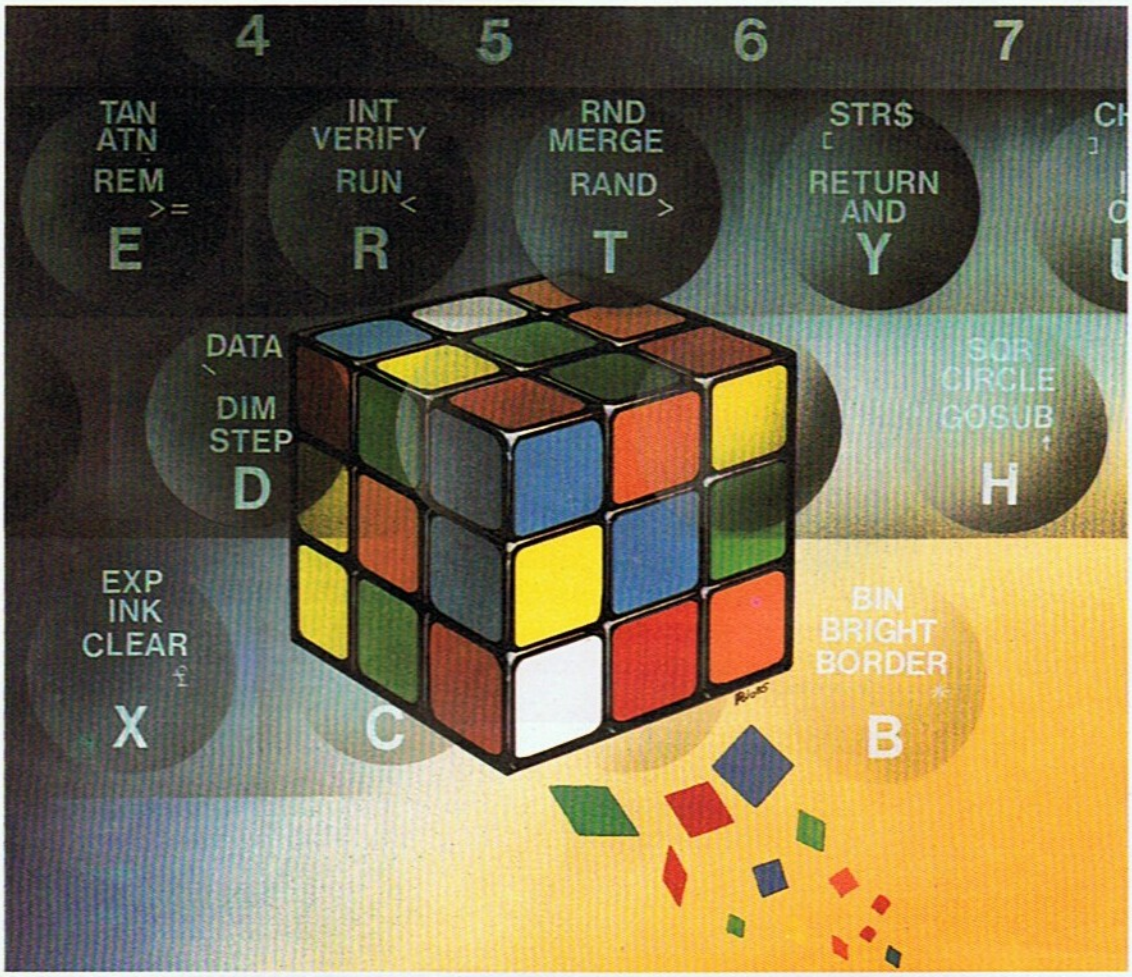
*

Al introducir el programa, no olvidemos sustituir los caracteres subrayados, por los gráficos definidos correspondientes a las teclas indicadas.

*

A la hora de efectuar una jugada, hemos de introducir el número correspondiente a: tipo de movimiento, número de bloque y dirección en la cual dicho bloque va a desplazarse.

```
  10 REM ***********************
  20 REM * J.M.MAYORAL SERRANO *
  30 REM ***********************
  40 GO SUB 5000
  42 BORDER 0: PAPER 0: INK 9: POKE 23658,
8
  45 CLS : GO SUB 7100
  46 GO SUB 5125: GO TO 200
  50 REM  PRESENTACION PANTALLA
  60 FOR N=1 TO 4
  70 FOR F=8 TO 12 STEP 2
  80 PRINT  INK N;AT F,7*(N-1)+1;"ABABAB"
  90 PRINT  INK N;AT F+1,7*(N-1)+1;"CDCDCD

 100 NEXT F
 110 NEXT N
 120 FOR F=1 TO 5 STEP 2
 130 PRINT  INK 5;AT F,8;"ABABAB"
 140 PRINT  INK 5;AT F+1,8;"CDCDCD"
 150 NEXT F
 160 FOR F=15 TO 19 STEP 2
 170 PRINT  INK 6;AT F,8;"ABABAB"
 180 PRINT  INK 6;AT F+1,8;"CDCDCD"
 190 NEXT F
 195 GO TO 370
 200 REM DIBUJO LIMITES CUBO
 205 INK 9
 210 PLOT 3,59: DRAW 225,0
 215 DRAW 0,56
 220 PLOT 3,60: DRAW 224,0
 225 DRAW 0,56
 230 PLOT 3,116: DRAW 225,0
 240 PLOT 3,115: DRAW 225,0
 250 PLOT 3,59: DRAW 0,56
 260 PLOT 4,59: DRAW 0,56
 270 PLOT 59,3: DRAW 57,0
 280 DRAW 0,169
 290 DRAW -56,0
 300 DRAW 0,-168
 310 PLOT 60,4: DRAW 55,0
 320 DRAW 0,167
 330 DRAW -54,0
 340 DRAW 0,-167
 350 PLOT 172,61: DRAW 0,54
 360 PLOT 171,61: DRAW 0,54
 365 GO TO 50
 370 REM INIC. VARIABLES
```




```
 375 DIM Z(3,3): DIM R(3,3)
 380 DIM M(3,3): DIM V(3,3)
 385 DIM C(3,3): DIM A(3,3)
 390 DIM P(3): LET CONTA=0
 395 GO SUB 6770: GO SUB 7137
 699 REM INTROD. DATOS
 700 PRINT AT 17,16; FLASH 1; PAPER 2;'MOV
IMIENTO?'
 705 FOR N=1 TO 3
 710 PRINT  PAPER 6;AT 19,17+N*2;N
 715 NEXT N
 720 LET K$=INKEY$
 725 IF CODE K$<49 OR CODE K$>51 THEN  GO
TO 720
 730 BEEP .2,40: BEEP .2,38
 735 LET MOV=VAL K$
 740 GO SUB 5195
 745 PRINT AT 17,17; FLASH 1; PAPER 4;' BL
OQUE ?'
 750 FOR N=1 TO 3
 755 PRINT  PAPER 7;AT 19,17+N*2;N
 760 NEXT N
 765 LET K$=INKEY$
 770 IF CODE K$<49 OR CODE K$>51 THEN  GO
TO 765
 775 BEEP .2,40: BEEP .2,38
 780 LET BLOCK=VAL K$
 785 GO SUB 5195
 790 PRINT AT 17,17; FLASH 1; PAPER 5;'DIR
ECCION ?'
 795 FOR N=1 TO 2
 800 PRINT  PAPER 2;AT 19,19+N*2;N
 805 NEXT N
 810 LET K$=INKEY$
 815 IF CODE K$<49 OR CODE K$>50 THEN  GO
TO 810
 820 BEEP .2,40: BEEP .2,38
 825 LET DIR=VAL K$
 830 GO SUB 5195
 835 LET FORM=4-BLOCK
 840 REM  PROGRAMA PRINCOPAL
 845 LET CONTA=CONTA+1
 850 GO SUB 6980
 900 GO SUB 6860
 910 GO SUB 6830
 920 GO TO 700
5000 REM  GRAFICOS USUARIO
5010 DATA 0,127,127,127,127,127,127,127
5020 DATA 0,254,254,254,254,254,254,254
5030 DATA 127,127,127,127,127,127,127,0
5040 DATA 254,254,254,254,254,254,254,0
5042 DATA 74,32,77,32,77,65,89,79,82,65,76
5043 DATA 32,83,69,82,82,65,78,79
5060 LET A$='ABCD'
5070 FOR N=1 TO LEN A$
5080 FOR K=0 TO 7
5090 READ A
5100 POKE USR A$(N)+K,A
5110 NEXT K
5120 NEXT N
5122 RETURN
5125 CLS : PRINT AT 0,0: PRINT
5130 FOR N=0 TO 18
5140 READ B
5150 PRINT  INK 5; BRIGHT 1;TAB 30;CHR$ B
5160 NEXT N
5170 PRINT  INK 5: BRIGHT 0;AT 2,31;'.'
5180 PRINT  INK 5;AT 4,31;'.'
5190 RETURN
5195 REM  SBR. BORRADO PARCIAL
5200 FOR F=17 TO 19 STEP 2
5205 FOR C=16 TO 28
5210 PRINT AT F,C;' '
5215 NEXT C
5220 NEXT F
5225 RETURN
5226 REM
5230 REM   SBR - 1
5231 REM
5235 REM PASO A PUENTE
5240 FOR M=1 TO 3
5245 LET P(M)=A(M,BLOCK)
5250 NEXT M
5255 REM PASO V a A (3 a 4)
5260 LET X=3
5265 FOR M=1 TO 3
5270 LET A(M,BLOCK)=V(X,FORM)
5275 LET X=X-1
5280 NEXT M
5285 REM PASO C a V (2 a 3)
5290 LET X=3
5295 FOR M=1 TO 3
5300 LET V(M,FORM)=C(X,BLOCK)
5305 LET X=X-1
5310 NEXT M
5315 REM PASO R a C (1 a 2)
5320 FOR M=1 TO 3
5325 LET C(M,BLOCK)=R(M,BLOCK)
5330 NEXT M
5331 REM PASO PUENTE a R
5332 FOR M=1 TO 3
5333 LET R(M,BLOCK)=P(M)
5334 NEXT M
5335 RETURN
5340 REM   SBR - 2
5345 REM MUEVE VERTICES
5350 LET P(1)=Z(1,3)
5355 LET Z(1,3)=Z(3,3)
5360 LET Z(3,3)=Z(3,1)
5365 LET Z(3,1)=Z(1,1)
5370 LET Z(1,1)=P(1)
5375 REM MUEVE ADYACENTES CENTR
5380 LET P(1)=Z(2,3)
5385 LET Z(2,3)=Z(3,2)
5390 LET Z(3,2)=Z(2,1)
5395 LET Z(2,1)=Z(1,2)
5400 LET Z(1,2)=P(1)
5405 RETURN
5410 REM   SBR - 3
5415 REM MUEVE VERTICES
5420 LET P(1)=M(1,3)
5425 LET M(1,3)=M(1,1)
5430 LET M(1,1)=M(3,1)
5435 LET M(3,1)=M(3,3)
5440 LET M(3,3)=P(1)
5445 REM MUEVE ADYACENTES CENTR
5450 LET P(1)=M(2,3)
5455 LET M(2,3)=M(1,2)
5460 LET M(1,2)=M(2,1)
5465 LET M(2,1)=M(3,2)
5470 LET M(3,2)=P(1)
5475 RETURN
5480 REM   SBR - 4
5485 REM
5490 FOR M=1 TO 3
5495 LET P(M)=C(M,BLOCK)
5500 NEXT M
5505 REM
5507 LET X=3
5510 FOR M=1 TO 3
5515 LET C(M,BLOCK)=V(X,FORM)
5520 LET X=X-1
5525 NEXT M
5530 REM
5535 LET X=3
5540 FOR M=1 TO 3
5545 LET V(M,FORM)=A(X,BLOCK)
5550 LET X=X-1
5555 NEXT M
5560 REM
5565 FOR M=1 TO 3
5570 LET A(M,BLOCK)=R(M,BLOCK)
5575 NEXT M
5580 REM
5585 FOR M=1 TO 3
5590 LET R(M,BLOCK)=P(M)
5595 NEXT M
5600 RETURN
5605 REM   SBR - 5
5610 REM
5615 LET P(1)=Z(1,3)
5620 LET Z(1,3)=Z(1,1)
5625 LET Z(1,1)=Z(3,1)
5630 LET Z(3,1)=Z(3,3)
5635 LET Z(3,3)=P(1)
5640 REM
5645 LET P(1)=Z(2,3)
5650 LET Z(2,3)=Z(1,2)
5655 LET Z(1,2)=Z(2,1)
5660 LET Z(2,1)=Z(3,2)
5665 LET Z(3,2)=P(1)
5670 RETURN
5675 REM   SBR - 6
5680 REM
5685 LET P(1)=M(1,3)
5690 LET M(1,3)=M(3,3)
5695 LET M(3,3)=M(3,1)
5700 LET M(3,1)=M(1,1)
5705 LET M(1,1)=P(1)
5710 REM
5715 LET P(1)=M(2,3)
5720 LET M(2,3)=M(3,2)
5725 LET M(3,2)=M(2,1)
5730 LET M(2,1)=M(1,2)
5735 LET M(1,2)=P(1)
5740 RETURN
5745 REM   SBR - 7
5750 REM
```

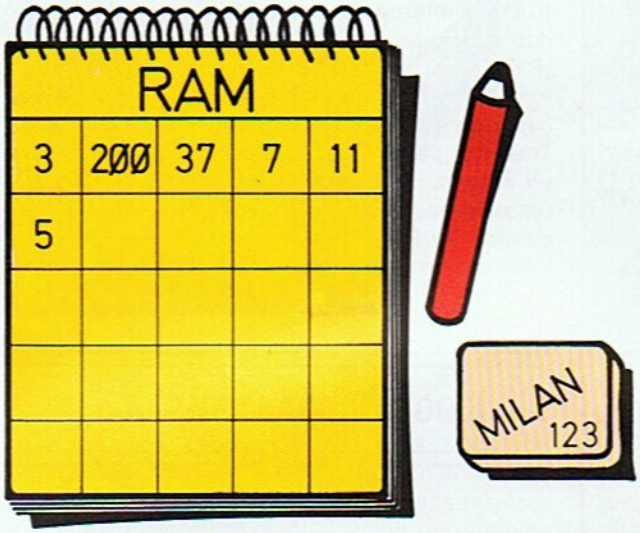
```
5755 FOR M=1 TO 3
5760 LET P(M)=Z(M,FORM)
5765 NEXT M
5770 REM
5775 LET X=1
5780 FOR M=1 TO 3
5785 LET Z(M,FORM)=A(BLOCK,X)
5790 LET X=X+1
5795 NEXT M
5800 REM
5805 LET X=3
5810 FOR M=1 TO 3
5815 LET A(BLOCK,M)=M(X,BLOCK)
5820 LET X=X-1
5825 NEXT M
5830 REM
5835 LET X=1
5840 FOR M=1 TO 3
5845 LET M(M,BLOCK)=C(FORM,X)
5850 LET X=X+1
5855 NEXT M
5860 REM
5865 LET X=3
5870 FOR M=1 TO 3
5875 LET C(FORM,M)=P(X)
5880 LET X=X-1
5885 NEXT M
5890 RETURN
5895 REM   SBR - 8
5900 LET P(1)=R(1,1)
5905 LET R(1,1)=R(3,1)
5910 LET R(3,1)=R(3,3)
5915 LET R(3,3)=R(1,3)
5920 LET R(1,3)=P(1)
5925 REM
5930 LET P(1)=R(1,2)
5935 LET R(1,2)=R(2,1)
5940 LET R(2,1)=R(3,2)
5945 LET R(3,2)=R(2,3)
5950 LET R(2,3)=P(1)
5955 RETURN
5960 REM   SBR - 9
5965 LET P(1)=V(1,1)
5970 LET V(1,1)=V(1,3)
5975 LET V(1,3)=V(3,3)
5980 LET V(3,3)=V(3,1)
5985 LET V(3,1)=P(1)
5990 REM
5995 LET P(1)=V(1,2)
6000 LET V(1,2)=V(2,3)
6005 LET V(2,3)=V(3,2)
6010 LET V(3,2)=V(2,1)
6015 LET V(2,1)=P(1)
6020 RETURN
6025 REM   SBR - 10
6030 REM
6035 FOR M=1 TO 3
6040 LET P(M)=M(M,BLOCK)
6045 NEXT M
6050 REM
6055 LET X=3
6060 FOR M=1 TO 3
6065 LET M(M,BLOCK)=A(BLOCK,X)
6070 LET X=X-1
6075 NEXT M
6080 REM
6085 LET X=1
6090 FOR M=1 TO 3
6095 LET A(BLOCK,M)=Z(X,FORM)
6100 LET X=X+1
6105 NEXT M
6110 REM
6115 LET X=3
6120 FOR M=1 TO 3
6125 LET Z(M,FORM)=C(FORM,X)
6130 LET X=X-1
6135 NEXT M
6140 REM
6145 FOR M=1 TO 3
6150 LET C(FORM,M)=P(M)
6155 NEXT M
6160 RETURN
6165 REM   SBR - 11
6170 LET P(1)=R(1,1)
6175 LET R(1,1)=R(1,3)
6180 LET R(1,3)=R(3,3)
6185 LET R(3,3)=R(3,1)
6190 LET R(3,1)=P(1)
6195 REM
6200 LET P(1)=R(1,2)
6205 LET R(1,2)=R(2,3)
6210 LET R(2,3)=R(3,2)
6215 LET R(3,2)=R(2,1)
6220 LET R(2,1)=P(1)
6225 RETURN
6230 REM   SBR -12
6235 LET P(1)=V(1,1)
6240 LET V(1,1)=V(3,1)
6245 LET V(3,1)=V(3,3)
6250 LET V(3,3)=V(1,3)
6255 LET V(1,3)=P(1)
6260 REM
6265 LET P(1)=V(1,2)
6270 LET V(1,2)=V(2,1)
6275 LET V(2,1)=V(3,2)
6280 LET V(3,2)=V(2,3)
6285 LET V(2,3)=P(1)
6290 RETURN
6295 REM   SBR - 13
6300 FOR M=1 TO 3
6305 LET P(M)=V(FORM,M)
6310 NEXT M
6315 REM
6320 FOR M=1 TO 3
6325 LET V(FORM,M)=M(FORM,M)
6330 NEXT M
6335 REM
6340 FOR M=1 TO 3
6345 LET M(FORM,M)=R(FORM,M)
6350 NEXT M
6355 REM
6360 FOR M=1 TO 3
6365 LET R(FORM,M)=Z(FORM,M)
6370 NEXT M
6375 REM
6380 FOR M=1 TO 3
6385 LET Z(FORM,M)=P(M)
6390 NEXT M
6395 RETURN
6400 REM   SBR - 14
6405 LET P(1)=A(1,1)
6410 LET A(1,1)=A(3,1)
6415 LET A(3,1)=A(3,3)
6425 LET A(3,3)=A(1,3)
6430 LET A(1,3)=P(1)
6435 REM
6440 LET P(1)=A(1,2)
6445 LET A(1,2)=A(2,1)
6450 LET A(2,1)=A(3,2)
6455 LET A(3,2)=A(2,3)
6460 LET A(2,3)=P(1)
6465 RETURN
6470 REM   SBR - 15
6475 LET P(1)=C(1,1)
6480 LET C(1,1)=C(1,3)
6485 LET C(1,3)=C(3,3)
6490 LET C(3,3)=C(3,1)
6495 LET C(3,1)=P(1)
6500 REM
6505 LET P(1)=C(1,2)
6510 LET C(1,2)=C(2,3)
6515 LET C(2,3)=C(3,2)
6520 LET C(3,2)=C(2,1)
6525 LET C(2,1)=P(1)
6530 RETURN
6535 REM   SBR - 16
6540 FOR M=1 TO 3
6545 LET P(M)=Z(FORM,M)
6550 NEXT M
6555 REM
6560 FOR M=1 TO 3
6565 LET Z(FORM,M)=R(FORM,M)
6570 NEXT M
6575 REM
6580 FOR M=1 TO 3
6585 LET R(FORM,M)=M(FORM,M)
6590 NEXT M
6595 REM
6600 FOR M=1 TO 3
6605 LET M(FORM,M)=V(FORM,M)
6610 NEXT M
6615 REM
6620 FOR M=1 TO 3
6625 LET V(FORM,M)=P(M)
6630 NEXT M
6635 RETURN
6640 REM   SBR - 17
6645 LET P(1)=A(1,1)
6650 LET A(1,1)=A(1,3)
6655 LET A(1,3)=A(3,3)
6660 LET A(3,3)=A(3,1)
6665 LET A(3,1)=P(1)
6670 REM
6675 LET P(1)=A(1,2)
6680 LET A(1,2)=A(2,3)
6685 LET A(2,3)=A(3,2)
6690 LET A(3,2)=A(2,1)
6695 LET A(2,1)=P(1)
6700 RETURN
6705 REM   SBR - 18
6710 LET P(1)=C(1,1)
6715 LET C(1,1)=C(3,1)
6720 LET C(3,1)=C(3,3)
6725 LET C(3,3)=C(1,3)
6730 LET C(1,3)=P(1)
6735 REM
6740 LET P(1)=C(1,2)
6745 LET C(1,2)=C(2,1)
6750 LET C(2,1)=C(3,2)
6755 LET C(3,2)=C(2,3)
6760 LET C(2,3)=P(1)
6765 RETURN
6770 REM  INIC. TABLAS
6775 FOR P=1 TO 3
6780 FOR Q=1 TO 3
6785 LET Z(P,Q)=1
6790 LET R(P,Q)=2
6795 LET M(P,Q)=3
6800 LET V(P,Q)=4
6805 LET C(P,Q)=5
6810 LET A(P,Q)=6
6815 NEXT Q
6820 NEXT P
6825 RETURN
6830 REM  CUBO COORECTO ?
6855 GO SUB 7900
6857 GO TO 700
6860 REM  DISPLAY CARAS/CUBO
6865 FOR F=1 TO 3
6870 FOR C=1 TO 3
6875 PRINT  INK Z(F,C);AT 2*F+6,2*C-1;"AB"
6880 PRINT  INK Z(F,C);AT 2*F+7,2*C-1;"CD"
6885 PRINT  INK R(F,C);AT 2*F+6,2*C+6;"AB"
6890 PRINT  INK R(F,C);AT 2*F+7,2*C+6;"CD"
6895 PRINT  INK M(F,C);AT 2*F+6,2*C+13;"AB
"
6900 PRINT  INK M(F,C);AT 2*F+7,2*C+13;"CD
"
6905 PRINT  INK V(F,C);AT 2*F+6,2*C+20;"AB
"
6910 PRINT  INK V(F,C);AT 2*F+7,2*C+20;"CD
"
6915 NEXT C
6920 NEXT F
6925 FOR F=1 TO 3
6940 FOR C=1 TO 3
6945 PRINT  INK C(F,C);AT 2*F-1,2*C+6;"AB"
6950 PRINT  INK C(F,C);AT 2*F,2*C+6;"CD"
6955 PRINT  INK A(F,C);AT 2*F+13,2*C+6;"AB
"
6960 PRINT  INK A(F,C);AT 2*F+14,2*C+6;"CD
"
6965 NEXT C
6970 NEXT F
6975 RETURN
6980 REM  SBR CONTROLA MOVIMIEN.
6985 IF MOV=1 AND BLOCK=1 AND DIR=1 THEN
GO SUB 5230: GO SUB 5340: RETURN
6990 IF MOV=1 AND BLOCK=1 AND DIR=2 THEN
GO SUB 5480: GO SUB 5605: RETURN
7000 IF MOV=1 AND BLOCK=2 AND DIR=1 THEN
GO SUB 5230: RETURN
7005 IF MOV=1 AND BLOCK=2 AND DIR=2 THEN
GO SUB 5480: RETURN
7010 IF MOV=1 AND BLOCK=3 AND DIR=1 THEN
GO SUB 5230: GO SUB 5410: RETURN
7015 IF MOV=1 AND BLOCK=3 AND DIR=2 THEN
GO SUB 5480: GO SUB 5675: RETURN
7020 REM
7025 REM
7030 IF MOV=2 AND BLOCK=1 AND DIR=1 THEN
GO SUB 5750: GO SUB 5895: RETURN
7035 IF MOV=2 AND BLOCK=1 AND DIR=2 THEN
GO SUB 6025: GO SUB 6165: RETURN
7040 IF MOV=2 AND BLOCK=2 AND DIR=1 THEN
GO SUB 5750: RETURN
7045 IF MOV=2 AND BLOCK=2 AND DIR=2 THEN
GO SUB 6025: RETURN
7050 IF MOV=2 AND BLOCK=3 AND DIR=1 THEN
GO SUB 5750: GO SUB 5960: RETURN
7055 IF MOV=2 AND BLOCK=3 AND DIR=2 THEN
GO SUB 6025: GO SUB 6230: RETURN
7060 REM
7065 REM
7070 IF MOV=3 AND BLOCK=1 AND DIR=1 THEN
GO SUB 6295: GO SUB 6400: RETURN
7075 IF MOV=3 AND BLOCK=1 AND DIR=2 THEN
GO SUB 6535: GO SUB 6640: RETURN
7080 IF MOV=3 AND BLOCK=2 AND DIR=1 THEN
GO SUB 6295: RETURN
7085 IF MOV=3 AND BLOCK=2 AND DIR=2 THEN
GO SUB 6535: RETURN
7090 IF MOV=3 AND BLOCK=3 AND DIR=1 THEN
GO SUB 6295: GO SUB 6470: RETURN
7095 IF MOV=3 AND BLOCK=3 AND DIR=2 THEN
GO SUB 6535: GO SUB 6705: RETURN
7100 REM NIVEL DE DIFICULTAD
7105 CLS
7110 PRINT AT 10,2; PAPER 6; INK 1;"INTROD
UCE NIVEL DE DIFICULTAD"
7115 PRINT AT 12,11;"1  -a-  9"
7120 LET K$=INKEY$
7125 IF CODE K$<49 OR CODE K$>57 THEN  GO
TO 7120
7130 BEEP .2,40
7135 LET NI=2*VAL K$: RETURN
7137 REM DESORDENACION
7138 PRINT  FLASH 1;AT 2,15; PAPER 2;"
                    ";AT 4,15;"
                    "
7139 PRINT AT 3,15; PAPER 2; FLASH 1;" DES
ORDENANDO "
7140 FOR G=1 TO NI
7145 LET MOV=INT (RND*3)+1
7150 LET BLOCK=INT (RND*3)+1
7155 LET DIR=INT (RND*2)+1
7157 LET FORM=4-BLOCK
7160 BEEP .1,40: GO SUB 6900
7165 NEXT G
7170 BEEP .1,40: BEEP .1,40
7175 FOR F=2 TO 4
7180 PRINT AT F,15;"
7185 NEXT F
7190 GO SUB 6860
7195 RETURN
7900 REM  COMPROBACION
7910 LET C1=Z(1,1)
7911 LET C2=R(1,1)
7912 LET C3=M(1,1)
7913 LET C4=V(1,1)
7914 LET C5=C(1,1)
7915 LET C6=A(1,1)
7920 FOR F=1 TO 3
7930 FOR C=1 TO 3
7940 IF Z(F,C)=C1 AND R(F,C)=C2 AND M(F,C)
=C3 AND V(F,C)=C4 AND C(F,C)=C5 AND A(F,C)
=C6 THEN  GO TO 7990
7950 RETURN
7990 NEXT C
7995 NEXT F
8000 REM  CUBO ACERTADO "
8010 PRINT  FLASH 1; PAPER 7; INK 9;AT 1,1
5;"LO CONSEGUISTE"
8020 PRINT  PAPER 6; INK 2;AT 3,16;CONTA;
PAPER 0; INK 9;"  MOVIM."
8030 PRINT AT 16,15;"OTRA PARTIDA ?"
8040 PRINT AT 18,19;"(S/N)"
8050 IF INKEY$="S" THEN  RUN
8060 IF INKEY$="N" THEN  GO TO 10000
8070 GO TO 8050
```

LETRAS QUE SON NUMEROS

EN capítulos anteriores hemos visto, de forma general, cómo el ordenador es capaz de tratar diferentes tipos de datos para realizar con ellos operaciones complicadas de cálculo, segmentación o comparación. Lo que no hemos hecho, en ningún momento, ha sido penetrar en la estructura interna de los datos, es decir, describir la forma en que éstos se encuentran almacenados dentro del ordenador.

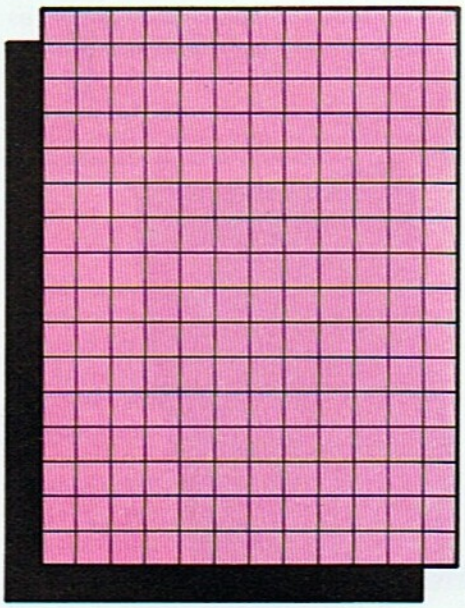
Simplificando mucho las cosas, podemos suponer que la memoria del ordenador es como una gigantesca hoja de papel cuadriculado, en la cual podemos borrar y escribir tantas veces como deseemos. Aún más, podemos elegir exactamente el lugar donde deseamos escribir, indicando el número correspondiente a una casilla determinada, siendo realmente ésta nuestra única restricción, puesto que nunca podremos escribir «a caballo» entre dos casillas.

Al conjunto de esta memoria se le denomina **RAM** *(Random Access Memory)*, cuya traducción es Memoria de Acceso Aleatorio. Con ello queremos decir que tenemos acceso a cualquier parte de ella, ya sea para leer su contenido o para alterarlo, escribiendo uno nuevo, con sólo reseñar su posición concreta dentro de la misma. Con una estructura similar a la descrita, el fabricante ha incluido también dentro de nuestro ordenador otra hoja de papel cuadriculado, pero ingeniándoselas de forma que podemos leer su contenido, pero no alterarlo. De hecho, podemos tratar de «escribir» sobre ella, siendo inútil nuestro esfuerzo por destruirla. A esta memoria de sólo lectura se le denomina **ROM** *(Read Only Memory)*.



En las memorias denominadas RAM, se admite tanto la lectura como la escritura de información.

Las memorias pueden considerarse como un papel cuadriculado, en el cual se encuentra la información.



En total, el Spectrum dispone de 65536 posiciones de memoria o cuadrículas, pertenecientes a uno u otro tipo de las memorias descritas; y en cada una de ellas puede almacenarse un carácter alfabético, numérico o cualquier símbolo. A cada una de estas posiciones descritas, se le denomina **BYTE** u **OCTETO.** Realmente, el nombre de **OCTETO** es el más descriptivo para nosotros: indica que cada una de las posiciones de memoria está compuesta a su vez por un conjunto de ocho unidades elementales, denominadas **BITS.**

Un **BIT** no es más que un impulso elemental binario, el cual puede adoptar dos únicos valores, a los cuales solemos asignar un significado lógico simbólico: SI o NO, VERDADERO o FALSO, y 1 ó 0. Aunque esta estructura puede parecernos en principio complicada de comprender, diremos

¡!

El código A.S.C.I.I. está compuesto por 128 caracteres, numerados del 0 al 127.

*

Para la definición de funciones del usuario, utilizamos **DEF FN,** seguido del nombre de la función y los parámetros (en caso de que existan), entre paréntesis y separados por comas.

¡!

El número total de caracteres son 256, puesto que corresponde con la máxima cantidad de información almacenable en un byte.

*

Al tipo de memoria que permite tanto la lectura como la alteración de contenido, se le denomina R.A.M. *(Random Access Memory).*

*

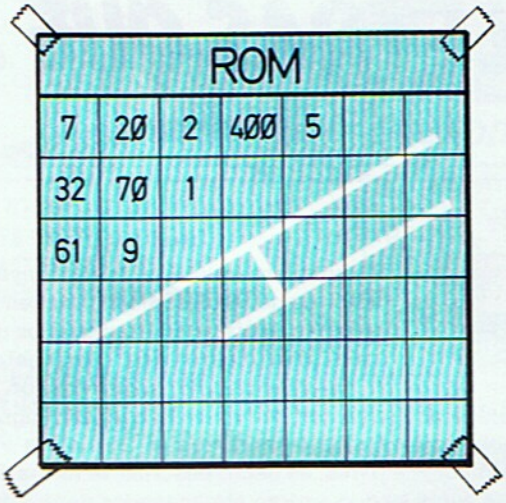
El Spectrum se ajusta casi totalmente al código A.S.C.I.I.

Al contrario que la RAM, la memoria ROM sólo admite operaciones de lectura.

que es, sin embargo, la única aceptable internamente por el ordenador.
En cualquier caso, no debe preocuparnos la estructura interna de almacenamiento del ordenador más que al nivel de posición de memoria **(BYTE)**, por ser ésta la mínima cantidad de información con sentido para nosotros, puesto que es capaz de albergar un carácter.
Tras este pequeño repaso, vamos a zambullirnos en el tema principal de este capítulo, no sin antes recordar que unas páginas antes, en la sección de TU SPECTRUM, podemos encontrar una información más amplia sobre las memorias, los bits y los bytes.

EL JUEGO DE CARACTERES

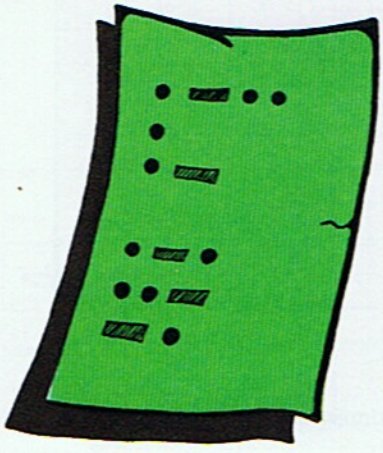
Es necesario que refresquemos ahora algunos conocimientos básicos sobre la aritmética binaria. Para empezar diremos que, del mismo modo que un dígito en base 10 (decimal) puede tener un contenido de 0 a 9, en base 8 (octal) puede albergar un número de 0 a 7, y en base 2 (binaria) únicamente los dígitos 0 y 1. Esto equivale a decir que, en cualquier base de numeración, los dígitos disponibles van desde el 0 hasta la base menos 1.
Si en base binaria sólo disponemos del 0 y el 1, con dos dígitos podemos albergar un número no superor a 3, con tres uno no superior a 7, con cuatro uno no superior a 15..., y con 8 un número que se encuentre en el rango de 0 a 255. Generalizando, podemos decir que siendo **N** el número de dígitos y **B** la base, se cumple la fórmula: **X=B↑N-1**. Donde **X** indica el máximo número que podemos almacenar.
Particularizando esta fórmula para **B=2** y para **N=8**, obtenemos el resultado de X=255. Este es precisamente el caso que nos ocupa, puesto que en un **BYTE** u **OCTETO** de memoria, podemos almacenar un número comprendido entre 0 y 255. Esta cantidad (256 posibilidades) corresponde con las dimensiones del juego de caracteres del



Spectrum, es decir, el conjunto de todas las letras, números, símbolos y palabras clave que encontramos en el teclado, e incluso algunos más, aunque no representables en la pantalla.

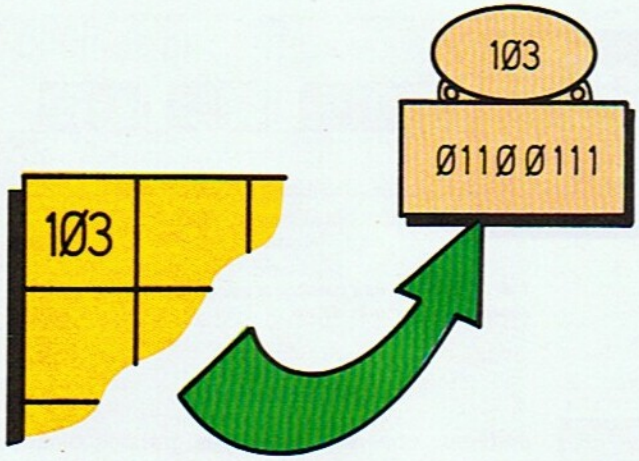
CODIFICACION DE CARACTERES

Como ya sabemos, al ordenador le es mucho más fácil llevar a cabo el tratamiento de números que de letras, y lo que es más, en realidad sólo puede efectuar un tratamiento numérico muy restringido, a base de unos y ceros. Sin embargo, a través de determinados sistemas de codificación, es capaz de llevar a cabo complejas operaciones matemáticas.

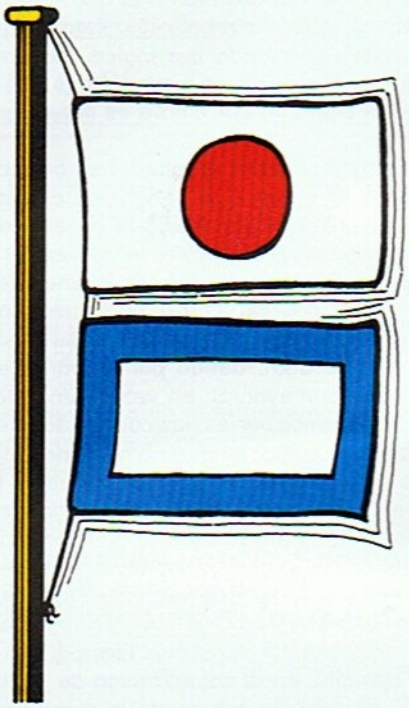


El sistema Morse sigue un tipo de codificación basado en los impulsos eléctricos.

Del mismo modo, el ordenador codifica los caracteres en forma de números, lo cual le permite dar el tratamiento adecuado a los caracteres, ya sean letras, símbolos, etc. Por otra parte, este proceso de codificación no es más que una adecuación al medio, que nos es bastante familiar. Así, cuando Sammuel Morse inventó su conocido código telegráfico, lo hizo obligado por la necesidad de sólo transmitir impulsos eléctricos de dos longitudes diferentes (puntos y rayas). Otra codificación similar, esta vez visual, es la que siguen las banderas de señalización en los barcos. Así pues, el hecho de codificar las letras por otro sistema, fácil de manejar por las herramientas de que se dispone, no es precisamente una de las originalidades del ordenador. En todo caso, vamos a estudiar la codificación que lleva a cabo nuestro Spectrum.



Aunque cada una de las posiciones de memoria contiene realmente 8 bits, a nosotros únicamente nos interesa el significado del byte completo.

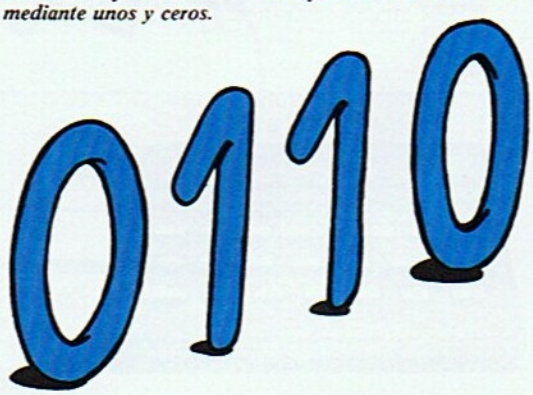


Las banderas de señalización en los barcos son un medio de codificación visual de información.

Desde la aparición de los ordenadores, se hizo necesaria la creación de un código estandarizado, para permitir la comunicación entre los diversos aparatos. De esta idea surgió el estándar de más frecuente utilización: el A.S.C.I.I. Estas siglas corresponden a *American Standard Code for Information Interchange,* que podríamos traducir como CODIGO NORMALIZADO AMERICANO PARA EL INTERCAMBIO DE INFORMACION; el mencionado código está compuesto por 128 caracteres, numerados del 0 al 127, que se detallan en el cuadro adjunto.

Como podemos ver, este incluye tanto letras mayúsculas como minúsculas, símbolos, signos de puntuación y caracteres de control; en definitiva, todos aquellos caracteres que pueden ser necesarios para cualquier comunicación. Sin embargo, aún quedan libres ciento veintiocho códigos (del 128 al 255). ¿Qué es lo que sucede con ellos? Simplemente son particulares de cada aparato, y se escapan al código estandarizado A.S.C.I.I. Así pues, cada fabricante de ordenadores los emplea para distintas finalidades; generalmente, contienen los gráficos y símbolos especiales. En el caso concreto del Spectrum, esta zona está reservada a los gráficos predefinidos y definibles, y a los *TOKEN,* es decir, las palabras BASIC que se componen de más de una letra.
El estudio del código A.S.C.I.I. tiene un cierto interés, puesto que como hemos dicho, es empleado por una gran cantidad de ordenadores y periféricos muy importantes, como por ejemplo, la mayoría de las impresoras. Fundamentalmente,

Todas las informaciones son codificadas en el ordenador mediante unos y ceros.

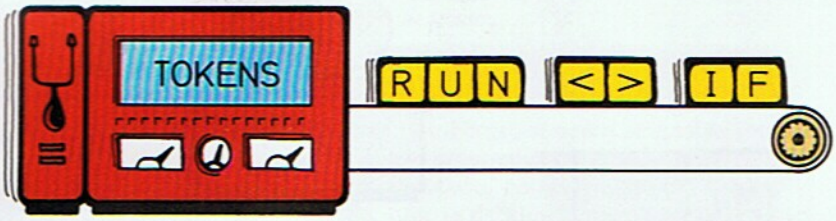


¡!

Los códigos del juego de caracteres que comprenden desde el 128 hasta el 255, se emplean para los gráficos y *TOKEN*s.

*

Un *TOKEN* es una palabra o símbolo BASIC completo, independientemente del número de caracteres individuales que lo compongan.



Un TOKEN es una palabra u operador BASIC compuesto por más de un carácter.

Para la comparación de caracteres, el ordenador se basa en la posición de los mismos dentro del código A.S.C.I.I.

*

La comparación de cadenas se efectúa comparando los caracteres que ocupan posiciones iguales en las cadenas a tratar.

*

La función de conversión de un carácter a su código es **CODE**.

podemos observar en él tres grandes bloques: El primero de ellos, se extiende desde el 0 hasta el 31, y está constituido por caracteres de control. En el caso concreto del Spectrum, sólo se utilizan los códigos 6 al 23, y de todas formas no se atienen totalmente al estándar A.S.C.I.I. A este bloque siguen una serie de símbolos y signos de puntuación, que comienzan por el código 32 (espacio), y finalizan en el 64, conocido generalmente por arroba (@). Dentro de este grupo, es importante destacar la presencia de los dígitos, que se extienden desde el 48 (el cero), hasta el 57 (el nueve).

El siguiente bloque importante comienza en el código 65 y llega hasta el 90, conteniendo todo el alfabeto inglés en mayúsculas. Una pequeña serie de tan solo seis caracteres, cuyos códigos van del 91 al 96, separan este bloque del último de importancia.

El último bloque a reseñar comprende los caracteres del 97 al 122, y representan de nuevo el alfabeto inglés, pero esta vez en minúsculas, seguido de los últimos caracteres especiales, los cuales abarcan desde el código 123 hasta el 127.

El juego de caracteres del ZX Spectrum, no respeta completamente la norma A.S.C.I.I. aunque sí en lo referente a los tres bloques fundamentales que hemos mencionado.

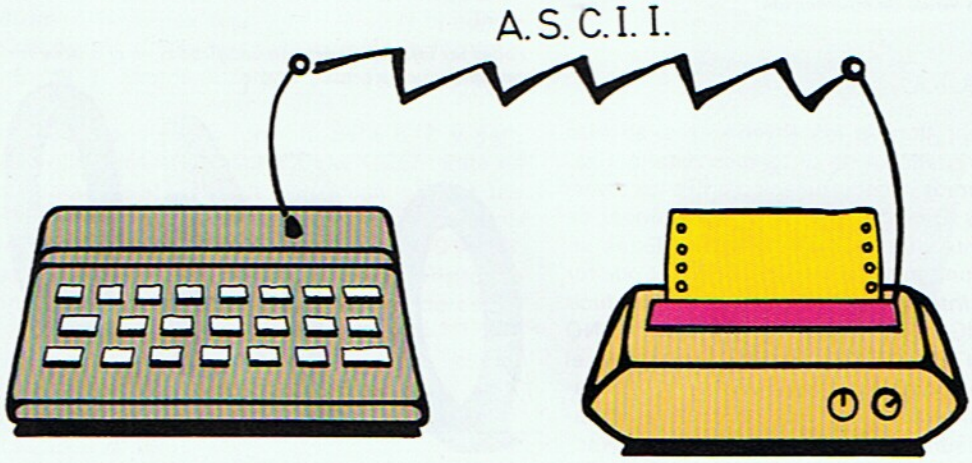
El hecho de que el ordenador utilice un código u otro, puede ser que nos parezca en estos momentos carente de importancia, pues se trata de un sistema de régimen interno que en teoría no nos afecta como programadores. No obstante, éste es un punto vital en el desarrollo de gran cantidad de programas, e inmediatamente conoceremos el porqué.

Cuando el ordenador efectúa comparaciones numéricas, su lógica resulta aplastante: "4 es mayor que 1"; sin embargo, no se inmuta en absoluto al realizar comparaciones entre letras, y se muestra bien seguro de que la «B» es mayor que la «A», y lo que es más, se obstina en el que el símbolo «%» es mayor que el símbolo «$». Esto último ya no nos está pareciendo tan lógico. Aceptamos que suponga que la «B» es mayor que la «A», pero ¿por qué el signo de por ciento es mayor que el de dólar?

El sistema que sigue el ordenador es bien claro, se basa en el orden que tienen los caracteres dentro del código ASCII (aunque no es exactamente este código, denominaremos así al empleado por el Spectrum). Así pues, cuando deseamos comparar dos caracteres cualquiera, nuestro Spectrum se limita a averiguar su posición dentro del A.S.C.I.I., dando por mayor a aquel cuyo código es mayor. Si en vez de un solo carácter comparamos series, las complicaciones no aumentan en exceso: se irán comparando los caracteres de las series que ocupen iguales posiciones; en el momento en que uno de los caracteres sea mayor que el otro, la comparación se decantará hacia ese lado. Por último, en el caso de que una de las series sea más corta que la otra, y todos los caracteres hayan sido iguales, se considerará mayor la serie más larga.

Así por ejemplo, en la comparación de "AHG" y "AHGA", se considerará mayor la segunda par-

El código más común para la comunicación entre ordenadores e impresoras es el A.S.C.I.I.



te, puesto que los primeros, segundos y terceros caracteres son iguales entre sí, pero sin embargo, la segunda cadena es más larga que la primera. Si la comparación se puede resolver antes del final de cualquiera de las cadenas, la longitud no se tiene en cuenta para nada; un ejemplo claro de ello, es el hecho de que la cadena «B» sea mayor que la «AAAAAAAA».

La comparación de cadenas se utiliza en gran cantidad de rutinas, como por ejemplo aquellas que manejan series, depuran entradas, etc... así pues, para atender las necesidades de programación sobre un tema que se ha revelado de vital importancia, el BASIC dispone de dos funciones de conversión directa, que relacionan al programador con el código que el ordenador emplea para sus caracteres. Estas dos funciones se denominan **CHR$** y **CODE**.

CHR$ Y CODE

La función **CHR$** tiene como argumento un número comprendido entre 0 y 255, proporcionando una cadena de salida con el carácter cuyo código se especifica en el argumento. El siguiente programa obtiene el juego completo de caracteres del Spectrum:

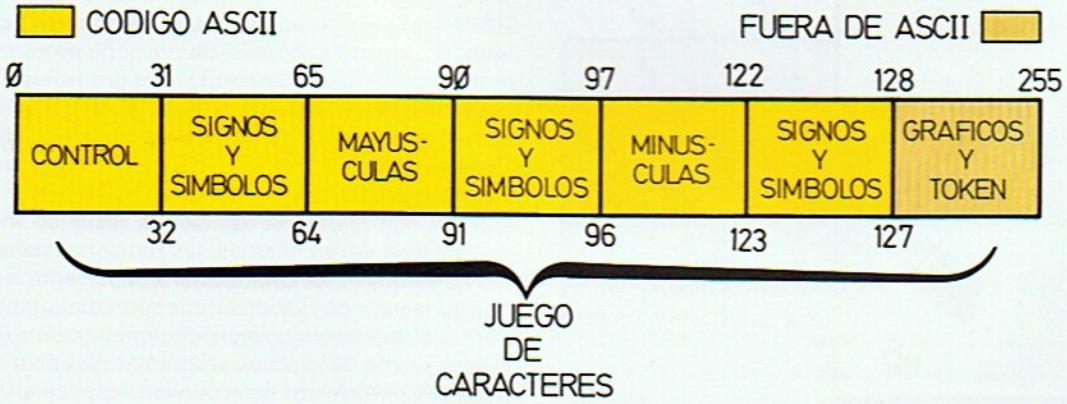
```
10 REM - JUEGO DE CARACTERES
20 PRINT "-------> CODIGOS ASCII
<--------"
30 FOR I=32 TO 127:PRINT CHR$ I;:NEXT
I
40 PRINT ""-----> CODIGOS RESTANTES
<------"
50 FOR I=128 TO 255:PRINT CHR$
I;:NEXT I
```

CODIGOS DE CONTROL

ABREVIATURA	SIGNIFICADO
NUL	Nulo.
SOH	Principio de encabezamiento.
STX	Comienzo de texto.
ETX	Fin de texto.
EOT	Fin de transmisión.
ENQ	Pregunta.
ACK	Acuse de recibo.
BEL	Timbre (señal).
BS	Retroceso.
HT	Tabulación horizontal.
LF	Cambio de renglón.
VT	Tabulación vertical.
FF	Página siguiente.
CR	Retroceso de carro.
SO	Fuera de código.
SI	En código.
SP	Espacio.
DLE	Encaje de trasmisión.
ESC	Escape.
DEL	Supresión.
DC1	Mando de dispositivo auxiliar 1.
DC2	Mando de dispositivo auxiliar 2.
DC3	Mando de dispositivo auxiliar 3.
DC4	Mando de dispositivo auxiliar 4.
NAK	Acuse de recibo negativo.
SYN	Sincronización de reposo.
ETB	Fin de bloque de transmisión.
CAN	Cancelación.
EM	Fin de medio físico.
SUB	Sustitución.
FS	Separador de fichero.
GS	Separador de grupo.
RS	Separador de registro.
US	Separador de unidad.

Al margen de su utilidad como función de codificación del juego de caracteres, **CHR$** tiene un gran interés en otro aspecto: la posibilidad de tra-

El juego de caracteres del Spectrum puede ser dividido en 7 zonas.



¡!

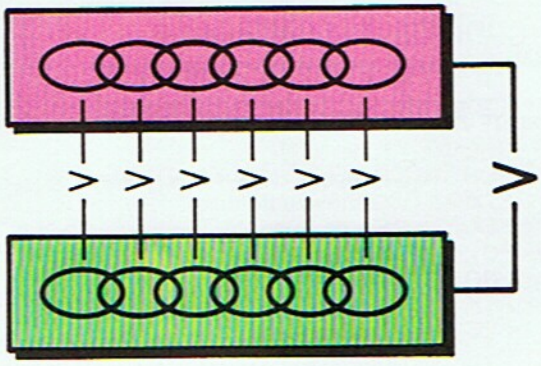
En el momento en que un carácter de una cadena es superior al correspondiente de la otra, la primera se considera mayor que la segunda.

*

Si no existe diferencia entre los primeros caracteres de dos cadenas, se considera mayor la más larga.

*

La función de conversión de un código en carácter es **CHR$**.



Las comparaciones entre cadenas, se efectúan entre los caracteres individuales que las componen, y que ocupan idénticas posiciones en cada una de ellas.

tar información de cadena como si fuese numérica, lo cual nos deja el camino abierto a las estructuras de bucle.

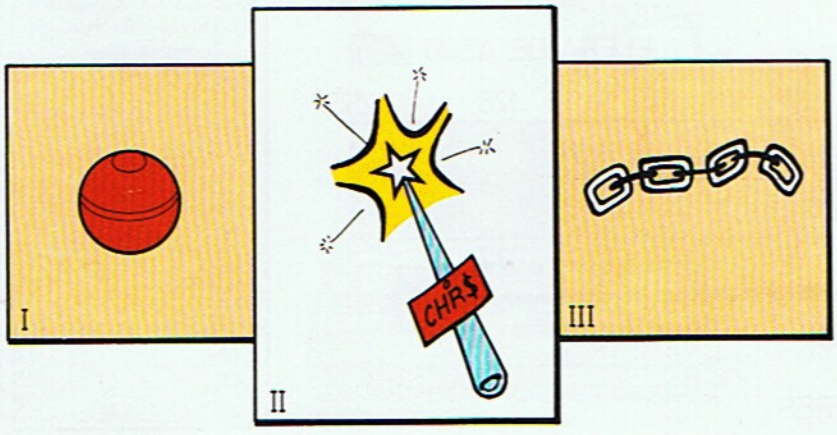
Supongamos que deseamos representar en columna el alfabeto inglés (mayúsculas). Nosotros reduciremos mentalmente esta tarea a una operación muy sencilla: «Para un carácter variando por orden desde la "A" hasta la "Z", escribamos cada vez el que corresponda».

Como ya sabemos, la traducción de este tipo de estructuras cíclicas al BASIC, se lleva a cabo de una forma prácticamente directa, mediante las sentencias **FOR-NEXT.** Así pues, estaremos tentados de programar la siguiente instrucción: **FOR I$="A" TO "Z": PRINT I$: NEXT I$.** Sin embargo, esta instrucción sería inmediatamente rechazada por el comprobador de sintaxis, puesto que no podemos utilizar variables de cadena como índices de ciclo.

La solución a nuestro problema consiste en encontrar una relación numérica entre los caracteres a representar. Esta bien puede ser el código que les corresponde dentro del juego de caracteres, y por tanto, será la función **CHR$** la que vendrá a sacarnos del apuro. La rutina anteriormente expuesta podría codificarse de la siguiente manera: **FOR I=65 TO 90: PRINT CHR$ I: NEXT I.** Esta instrucción se puede traducir al Castellano, diciendo: «para los códigos variando del 65 al 90, escribe el carácter cuyo código corresponde». La función **CODE** es inversa a **CHR$:** a partir de un argumento de cadena, produce el código numérico del carácter correspondiente. Más estrictamente diremos que obtiene el código del primer carácter de una cadena, devolviendo el valor 0 si la cadena es vacía. Así, obtendríamos el mismo resultado (65) en los dos supuestos siguientes: **PRINT CODE "A"** y **PRINT CODE "ABC".** También podemos comprobar que las funciones **CHR$** y **CODE** son inversas, escribiendo lo siguiente: **PRINT "A"** y **PRINT CHR$ CODE "A".**

Mediante DEF FN *los usuarios podemos definir nuestras propias funciones.*

La función CHR$ *nos permite obtener el carácter que corresponde a determinado código.*



El hecho de que **CHR$** opere en base a argumentos numéricos, no sólo reporta el beneficio de poder ser empleado en estructuras cíclicas, sustituyendo a los caracteres, sino que también le facilita el camino hacia las funciones definidas por el usuario. Efectivamente, hasta ahora, hemos estudiado funciones que el lenguaje BASIC incorpora, y proporcionan un determinado resultado; sin embargo, el BASIC dispone entre sus sentencias de una que aporta una gran flexibilidad en su programación, y lo que es más, una personalización de los programas: la posibilidad de que cada programador podamos definir nuestras propias funciones.

Aunque con ciertas restricciones respecto a la operatividad de las mismas, las funciones definidas contribuyen de una manera importante a la programación de calidad, aunque lamentablemente, el frecuente desconocimiento de su utilidad las aparta de muchos programas que podrían incluirlas, mejorando determinados aspectos.

DEFINICION DE FUNCIONES

La definición de funciones propiamente dicha, se canaliza a través de la sentencia **DEF FN** *(DEFine FuNction)* seguida del nombre de la función y los parámetros necesarios, entre paréntesis, y separados por comas si existe más de uno. De comienzo parece un poco lioso, pero pronto veremos que la realidad es otra; estudiemos separadamente cada uno de los componentes expuestos.

Los nombres de función deben constar de una sola letra, seguida o no del símbolo dólar **($)**, dependiendo del tipo de resultado que esperemos obtener de ella. De este modo, son funciones de cadena aquellas que terminan en dólar **($)**, y numéricas las demás.

Los parámetros pueden ser ninguno, uno, o más de uno, en cuyo caso deberán ir separados por comas, y siempre encerrados entre paréntesis a continuación del nombre de la función; incluso si no existen parámetros deberán indicarse los paréntesis, aunque sin ningún contenido. Estos parámetros representan el argumento de la función definida y, por tanto, el resultado de ésta dependerá directamente del valor de los mismos.

Los parámetros pueden ser de cualquier tipo (numéricos o de cadena), y no existe ningún problema para combinarlos en una misma función. La única restricción estriba en que los parámetros de cadena deben ser nombres válidos de variable de este tipo —una letra seguida del símbolo dólar **($)**—, y los numéricos, nombres de una sola letra. Como en el caso de las variables, es indiferente que el parámetro se escriba en mayúsculas o minúsculas; de esto se desprende que el máximo número de parámetros de un tipo que puede incluir una función son veintiséis (las letras del alfabeto), y en total cincuenta y dos (veintiséis de un tipo y veintiséis del otro).

Vamos a definir una función muy simple. La llamaremos **P**, y llevará a cabo el producto de dos números; así pues, la línea de definición será: **1 DEF FN P(X,Y)=X*Y**. Donde **X** e **Y** son los parámetros. Si nos damos cuenta, hemos antepuesto un número de línea a la definición, esto se debe a que esta sentencia opera de forma similar a **DATA**, y no puede ser empleada en el modo directo.

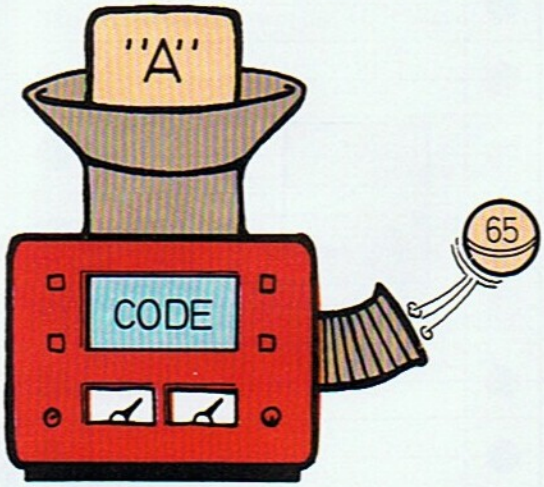
La forma en que ejecutamos una función definida es mediante la función BASIC **FN** *(FuNction)*, la cual lógicamente tiene una sintaxis muy similar a la definición: debe ir seguida del nombre de la función a ejecutar, y de los parámetros entre paréntesis. Como cualquier función BASIC, precisa de una palabra clave anterior que indique el destino del resultado. Por ejemplo, vamos a escribir el valor de **P** creada anteriormente, para **X=5** e **Y=7**. La instrucción será en este caso **PRINT FN P(5,7)**.

El resultado obtenido será 35, pero no es esto lo importante, sino que no hemos utilizado ninguna sentencia de asignación (**LET, INPUT** o **READ**), para otorgar valores a **X** e **Y**. Esto es de considerable importancia, puesto que de ello se desprende que no existe ninguna relación entre los parámetros de una función y unas posibles variables con el mismo nombre.

Esto no quiere decir que no se pueden utilizar variables dentro de la definición de una función, sino que, cuando se ha utilizado un determinado nombre como parámetro, el BASIC no considera

EL CODIGO ASCII

BITS 3210	BITS 654							
	000	001	010	011	100	101	110	111
0000	NUL	DLE	SP	0	@	P	£	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	"	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	'	7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	x
1001	HT	EM	)	9	I	Y	i	v
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[	k	{
1100	FF	FS	,	<	L	/	1	|
1101	CR	GS	–	=	M	]	m	}
1110	SO	RS	.	>	N	¡	n	ñ
1111	SI	US	/	?	O	—	o	DEL



CODE es la función BASIC que nos permite averiguar el código que corresponde a determinado carácter.

El resultado de una función definida depende de determinados parámetros, y es obtenido mediante la sentencia **FN.**

Al tipo de memoria que sólo admite la lectura, pero no la alteración de contenido, se le denomina R.O.M. *(Read Only Memory).*

*

Las siglas A.S.C.I.I. corresponden a *American Standard Code for Information Interchange,* cuya traducción es Código Normalizado Americano para el Intercambio de Información.

*

En el código A.S.C.I.I. podemos diferenciar tres bloques principales: caracteres de control, alfabeto en mayúsculas y alfabeto en minúsculas.

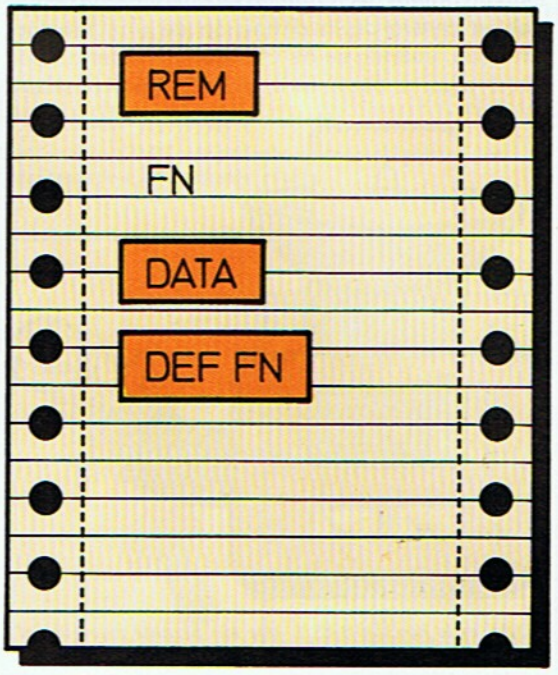
la existencia de ninguna variable con ese mismo nombre. De no ser así, es decir, en caso de que una variable, cuyo nombre no se emplea en los parámetros, sí tome parte en la definición de la función, su valor se obtendrá del que tuviera dicha variable en el momento de la ejecución de **FN.** Veamos un ejemplo:

```
10 DEF FN P(X,Y)=X*Y+A
20 LET A=15
30 PRINT FN P(2,3)
```

El resultado obtenido por esta función será 21 (2*3+15=21), puesto que los valores de **X** e **Y** se tomarán de los parámetros, y el de **A,** al no encontrarse en la definición dentro de los mismos, será extraído del área de variables. Por supuesto, no es necesario utilizar constantes al evaluar una función definida **(FN),** sino que podemos utilizar variables o expresiones. La única restricción es que figuren exactamente los parámetros precisos; en caso contrario, obtendremos el error **Q Parameter error** (Q Error de parámetro).

La utilidad de **DEF FN** es bastante grande, puesto que nos permite definir algunas funciones interesantes, presentes en otros BASIC, pero de los que carece el de Sinclair. Así por ejemplo, la función **MOD,** da como resultado el resto de la división entre dos números, y puede ser simulada mediante: **DEF FN M(A,B)=A-B*INT (A/B).**

Las sentencias **DEF FN,** *al igual que* **DATA** *y* **REM,** *son ignoradas por el ordenador durante la interpretación de la línea en que se encuentran.*



FN es la función BASIC que se emplea para acceder a los resultados de una función definida.

En el siguiente ejemplo, mostramos la función que permite tomar de la cadena **X$** los **X** caracteres de su derecha; esta función se denomina en otros BASIC **RIGHT$: DEF FN R$(A$,A)= A$(LEN A$-A+1 TO).** Como último ejemplo, podemos definir la función de redondeo a cualquier número de cifras, donde **X** es el número a redondear, e **Y** la cantidad de decimales requeridos: **DEF FN A(X,Y)=INT ((X+(5/10↑(Y+1)))* 10↑Y)/10↑Y.**

En definitiva, **DEF FN** es una herramienta de mucha utilidad, con la cual debemos practicar.

MANEJO DE SERIES

BAJO la denominación general de manejo de series, incluimos habitualmente tres actividades bien diferentes: la ordenación de series o *sort*, la busqueda en las mismas, y su desordenación o *SHUFFLING*. Anteriormente, nos ocupamos de la primera de estas tareas, y seguramente la más importante de todas ellas, puesto que, como veremos más adelante, en algunas ocasiones, antes de efectuar una búsqueda es conveniente someter a la serie a un proceso previo de *sort*. Por otra parte, la aplicación del *shuffling*, método totalmente opuesto al *sort*, se da casi únicamente en determinados programas de juegos; por tanto, en un ámbito bastante más restringido que los *sort*, debido a lo cual, no le dedicaremos la misma atención que a su antónimo.
En primer lugar, nos ocuparemos del estudio de las técnicas de búsqueda.

BUSQUEDA EN SERIES: SHUFFLINGS

El fin último de las técnicas de manejo de series es un acceso cómodo y rápido a la información que éstas contienen. En este sentido, debemos de procurar someter a las mismas a ciertas operaciones, encaminadas a la facilitación de la posterior labor de acceso. Estas operaciones suelen ser de clasificación, aunque como veremos más adelante existen determinadas ocasiones en que la serie de partida está ya ordenada y lo que se pretende es someterla a un *shuffling*, puesto que es conveniente que los valores a acceder se encuentren desordenados.
Una vez realizado el tratamiento de la serie, el acceso a cualquier elemento de ella depende del tipo de almacenamiento de la información escogido. En este caso, podemos hablar del acceso directo, o de la necesidad de una búsqueda. La primera circunstancia es la ideal, dado que aumenta al máximo la comodidad del acceso, y disminuye al mínimo el tiempo en conseguir el dato. En el acceso directo, utilizamos una simple herramienta BASIC para recuperar la información contenida en determinado punto de un colectivo (serie). Pongamos un ejemplo bien sencillo: la serie a tratar será toda la memoria, y el elemento a acceder el 32000. En este caso, el problema será bien fácil de resolver; sólo necesitamos ejecutar **PRINT PEEK 32000.** Esto ha sido posible, sólo porque la referencia del elemento a buscar coincidía con el índice por el cual se accede a la serie. Veamos otro ejemplo algo más práctico. Esta vez, en un juego en el cual se lanza un dado, nos interesa saber si determinada puntuación ha aparecido ya. Para ello podemos utilizar un simple sistema de banderas, consistente en hacer alguna variable, fácilmente accesible, igual a un valor indicador de la aparición anterior o no del elemento al cual representa. En este sentido, podemos dimensionar un vector numérico de seis elementos, los cuales contendrán inicialmente el valor 0. Asimismo, adoptaremos el convenio de cambiar el elemento correspondiente al valor a uno (o a cualquier valor distinto de cero), para señalar que dicho valor ya ha sido extraído.
A ninguno se nos escapa que este tipo de acceso sólo es posible con determinadas estructuras de datos, y sobre todo, presenta el inconveniente de la gran cantidad de memoria utilizada. Para solapar un poco este inconveniente, podemos emplear vectores de cadena, que como ya sabemos ocupan menos memoria que los numéricos, pero esto no supondrá una solución definitiva.
Una vez que nos vemos obligados a realizar una búsqueda, podemos optar por dos sistemas: el de búsqueda secuencial, y el de extrapolación de in-

¡!

Por *shufflings* entendemos el conjunto de técnicas que permiten, partiendo de una serie ordenada, obtener una serie distribuída aleatoriamente.

*

El método de búsqueda por extrapolación de intervalos supera en eficacia, por lo general, al secuencial, si bien es cierto que sólo puede ser aplicado sobre series ordenadas.

SORTS
(ordenacion)

BUSQUEDA

SHUFFLINGS
(desordenacion)

Bajo la denominación general de manejo de series, se incluyen fundamentalmente tres procesos: ordenación (sort), búsqueda, y desordenación (shuffling).

¡!

Fundamentalmente, existen dos métodos de búsqueda en series: secuencial y por extrapolación de intervalos.

La búsqueda secuencial funciona con series desordenadas, pero puede ser aumentada su eficacia si se aplica sobre listas ordenadas.

Intercambiar los valores de dos elementos, ya sean variables, componentes de una serie, etc... se denomina *swapping*.

Para efectuar correctamente un *swapping*, hemos de tener muy presente la necesidad de emplear una variable puente, en la cual poder almacenar temporalmente el valor de uno de los elementos.

Uno de los sistemas para eliminar un elemento de una serie, consiste en solaparlo mediante el desplazamiento de una posición, del resto de los componentes de la serie.

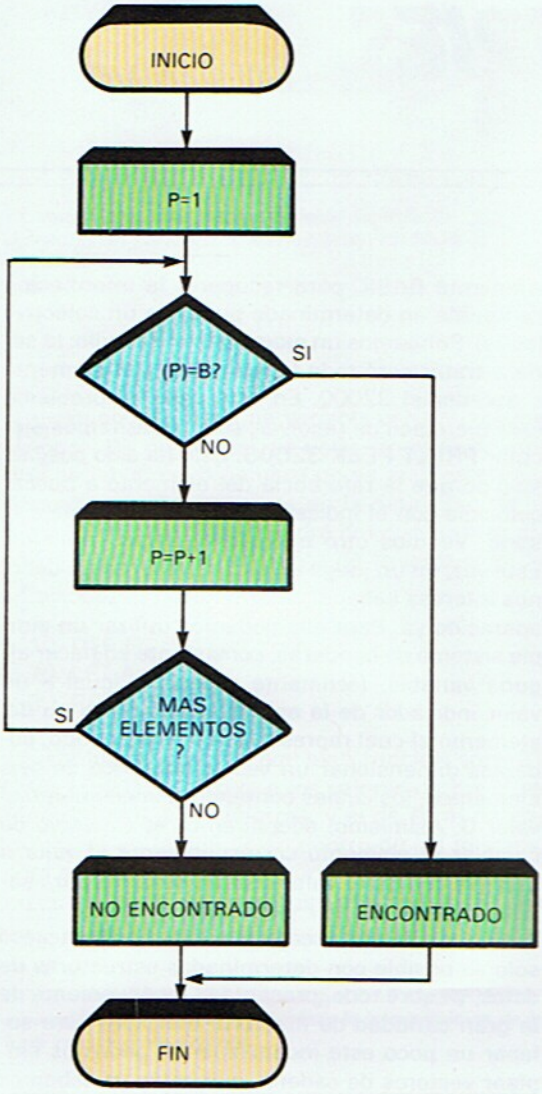


Diagrama de flujo de la búsqueda secuencial para series desordenadas.

tervalos. El primero de ellos es el más simple, y también el más ineficaz, si bien es cierto que el de extrapolación de intervalos no puede ser empleado sobre series desordenadas.

La búsqueda secuencial, como su propio nombre indica, consiste en ir analizando cada uno de los elementos de la serie, desde el primero hasta el último, siguiendo un orden de índices. De esta manera, estaremos seguros de localizar el elemento buscado, en caso de que exista. En el siguiente ejemplo, intentamos localizar un imaginario elemento **B**, en el vector **V (20)**:

```
10 FOR P=1 TO 20
20 IF V(P)=B THEN PRINT "ENCONTRADO EN ";P: STOP
30 NEXT P
40 PRINT "NO ENCONTRADO EN LA SERIE"
```

Si se dispone de la ventaja de una serie ordenada, el método de búsqueda secuencial puede mejorarse, evitando la necesidad de recorrer toda la serie para saber si el elemento existe. Suponiendo, por ejemplo, que la serie mantuviera un orden ascendente, sólo deberemos analizar los elementos, hasta el punto en el cual éstos sean superiores a **B**.

```
10 FOR P=1 TO 20
20 IF V(P)=B THEN PRINT "ENCONTRADO EN ";P: STOP
30 IF V(P)>B THEN GO TO 50
40 NEXT P
50 PRINT "NO ENCONTRADO EN LA SERIE"
```

El último sistema de búsqueda a estudiar es algo más complejo, y en la mayoría de los casos mucho más eficaz. Recordemos que presenta el inconveniente de sólo poder ser utilizado en series ordenadas, al igual que la segunda versión de nuestra rutina de búsqueda secuencial.

El método consiste en escoger el punto medio de la serie, y averiguar hacia qué mitad se encuentra el valor a localizar. La siguiente investigación considerará esta parte de la lista como el intervalo a estudiar, volviendo a realizar la división de éste en dos zonas. De esta manera, la longitud de la serie a estudiar se reduce cada vez a la mitad, con la seguridad de que, si existe, el elemento será localizado, y de no ser así, el programa se detendrá al alcanzar el intervalo la longitud cero.

En el ejemplo, utilizaremos la variable **LI** para señalar el límite inferior del intervalo, **LS** para el superior, **L** para la longitud del mismo, y **P** como puntero de mitad de serie (elemento a analizar). Como en el caso anterior, suponemos la búsqueda de un valor dado **B**, en la serie ordenada **V (20)**.

```
10 LET L=20: LET LI=0
20 LET L=INT (L/2+.5)
30 LET P=LI+L
40 IF B>V(P) THEN LET LI=P: GO TO 70
50 IF B<V(P) THEN GO TO 70
60 PRINT "ENCONTRADO EN ";P:STOP
70 IF L>1 THEN GO TO 20
80 PRINT "NO ENCONTRADO EN LA SERIE"
90 STOP
```

Tengamos en cuenta, que las rutinas de ejemplo anteriores no funcionarán directamente, dado que no se encuentra definido el valor a localizar **B**, ni tan siquiera la serie a tratar **V (20)**. Por otra parte, el tratamiento se ha particularizado en todos los casos para veinte elementos, aunque lógicamente este valor puede ser alterado a cualquiera que sea la longitud de la serie.

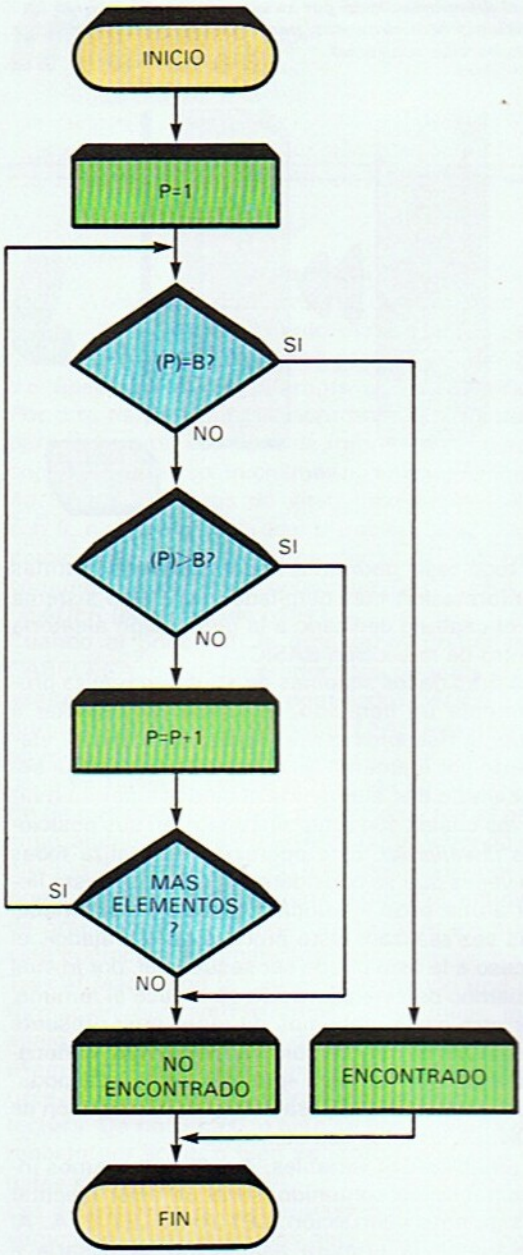
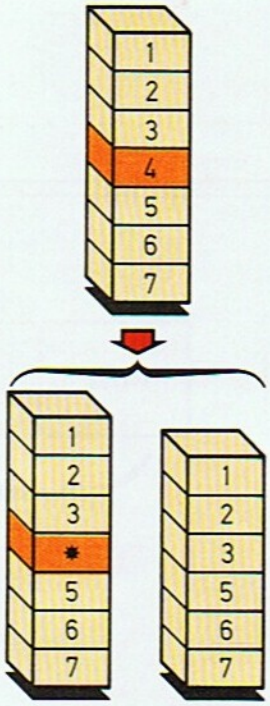


Diagrama de flujo de la búsqueda secuencial en una serie ordenada.

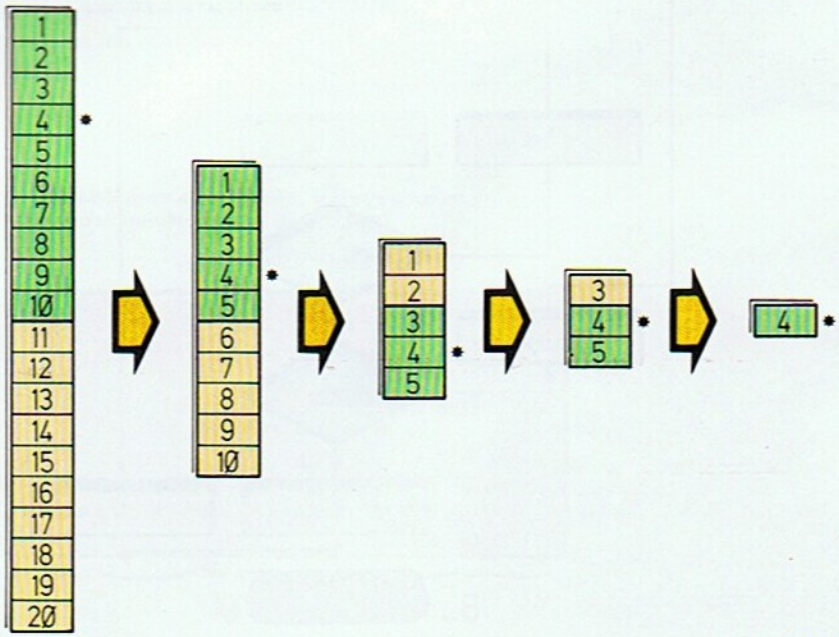
Una vez vistos los métodos de clasificación y búsqueda, nos ocuparemos por último de los *shuffling,* o sistemas de desordenación. Como ya hemos dicho, estos se emplean fundamentalmente en los juegos de azar que parten de series ordenadas; de hecho, su denominación inglesa (barajado), se deriva de la prácticamente ineludible utilización en los juegos de cartas.
Como en el caso de las búsquedas, comentaremos los dos sistemas de desordenación más ampliamente extendidos: el primero de ellos se basa en la extracción aleatoria de elementos, sin recurrir a la desordenación de la serie, debido a lo cual no se puede decir que sea exactamente un método de *shuffling,* aunque lo incluiremos en este epígrafe debido a que su efecto es similar; por el contrario, la desordenación por excelencia consiste en someter a la serie a un intercambio de elementos *(SWAPPING).*

En el sistema de extracción aleatoria, el elemento escogido es suprimido, o sustituido por un indicador significativo de la extracción.



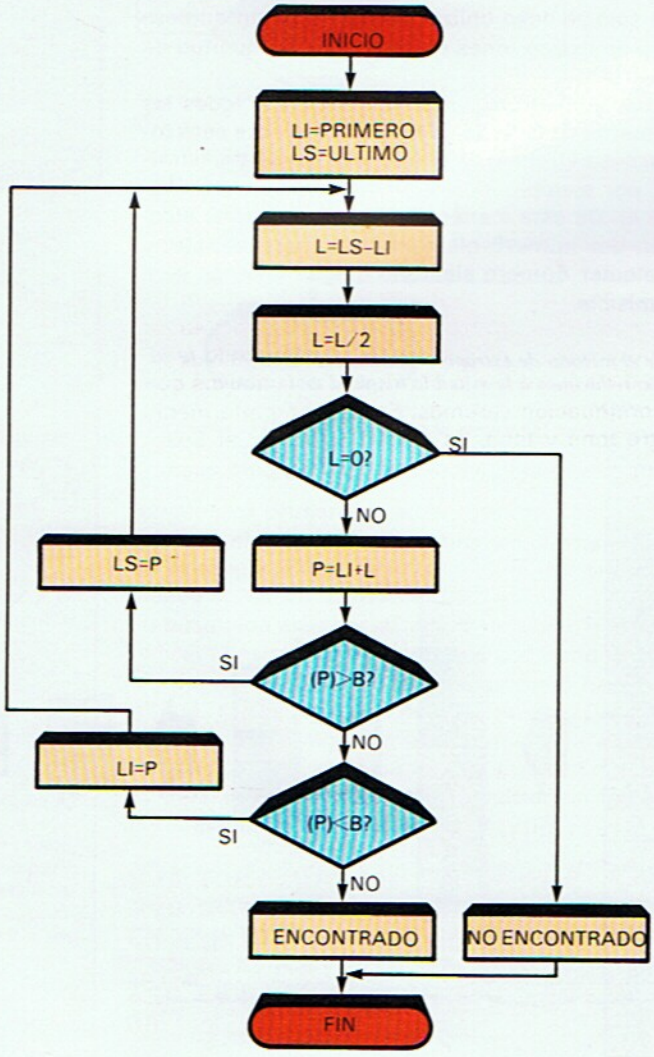
Siguiendo el sistema de extracción aleatoria, podemos optar por dos formas diferentes de tratar la serie una vez extraído determinado elemento: o bien se sustituye por una marca que señale la anterior extracción del mismo, o bien se elimina de la serie.
El primero de los métodos es en principio el más rápido, pero sin embargo, cuando aumenta el número de extracciones, va haciéndose más y más lento, puesto que aumentan las probabilidades de que el elemento extraído aleatoriamente haya aparecido con anterioridad. Por tanto, este sistema sólo se debe utilizar partiendo de un número bajo de extracciones en relación a la longitud de la serie.
Si por el contrario se pretende extraer todos los números de la serie, lo más conveniente será tomarnos el trabajo de eliminar el elemento extraído, por ejemplo, desplazando los superiores sobre él. De esta manera, el intervalo para la elección del número aleatorio se va reduciendo, y cualquier número aleatorio que se obtenga será admisible.

Por el método de extrapolación de intervalos, en cada paso reducimos a la mitad la longitud de la serie a investigar.

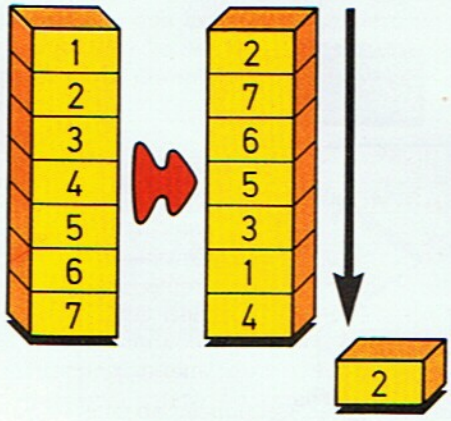


```
 10 DIM D(6)
 20 BORDER 4: PAPER 4: INK 0: BRIGHT 0:
FLASH 0: CLS
 30 PLOT 79,128: DRAW 97,0: DRAW 0,-97:
DRAW -97,0: DRAW 0,97
 40 PAPER 6: BRIGHT 1
 50 FOR I=6 TO 17: PRINT AT I,10;"
      ": NEXT I
 60 PRINT AT 6,10;"RESULTADO: 0";AT 8,1
3;"VECTOR";
 70 PRINT AT 8,13; OVER 1;"______"
 80 FOR I=1 TO 6: PRINT AT 9+I,14;I;"-
0": NEXT I
 90 LET R=INT (RND*6)+1
100 PRINT AT 6,21;R
110 IF D(R)=1 THEN  PRINT AT 17,10;"SI
HA SALIDO": GO TO 130
120 LET D(R)=1: PRINT AT 17,10;"NO HA S
ALIDO"
130 PRINT AT 9+R,14; OVER 1; INVERSE 1;
"     ";
140 PRINT AT 9+R,14; OVER 1; INVERSE 0;
"     "
150 IF INKEY$="" THEN  GO TO 130
160 PRINT AT 9+R,14;R;"- ";D(R)
170 FOR I=0 TO 9: NEXT I: GO TO 90
```

Diagrama de flujo del método de extrapolación de intervalos.



En el desordenamiento por swapping, se intercambian las posiciones de n elementos, para posteriormente proceder a su extracción secuencial.



En todo caso, podremos encontrar ciertas rutinas e información más detallada sobre este sistema en el capítulo dedicado a la generación aleatoria dentro de la sección BASIC.

El último de los sistemas de *shuffling* realiza propiamente un barajado, de forma muy similar a como lo hacemos las personas, pero eso sí, elemento por elemento. El método se basa en la selección de dos elementos cualquiera del intervalo, los cuales son intercambiados en sus posiciones *(swapping)*. Esta operación se realiza todas las veces que se consideren oportunas, hasta llegar a una serie lo suficientemente desordenada. Una vez realizado este proceso de «barajado», el acceso a la lista puede ser secuencial, por lo cual el tiempo de esta operación se reduce al mínimo.

Por otra parte, debemos de tener muy presente al realizar el intercambio de elementos, la necesidad de una variable «puente» en la que podamos almacenar temporalmente la información de cualquiera de los dos elementos.

Si tenemos dos variables, **A** y **B**, y deseamos intercambiar su contenido, sería un error ejecutar la siguiente instrucción: **LET A=B: LET B=A.** Al efectuarse la primera asignación, **A** pasaría a adoptar el contenido de **B**, perdiéndose su valor anterior. Por tanto, es necesaria la existencia de una variable «puente», la cual denominaremos **C** en el ejemplo, con la única misión de preservar el contenido de **A** contra la pérdida por la primera asignación.

Siguiendo este último sistema, formaremos una asignación «triangular» entre las variables **A, B** y **C:**

```
10 LET C=A: REM CONSERVA EN C EL VA-
LOR INICIAL DE A
20 LET A=B: REM DEPOSITA EN A EL VA-
LOR INICIAL DE B
30 LET B=C: REM DEPOSITA EN B EL VA-
LOR INICIAL DE A
```

INDEX

La mayoría de los microordenadores utilizan los casetes como unidad de almacenamiento masivo, tanto para datos como para programas; y nuestro Spectrum no es diferente en este sentido. Por otra parte, sigue siempre un mismo método para la lectura/escritura de información en este soporte magnético. Inicialmente, envía un pequeño bloque de datos de vital importancia, tales como: nombre para la identificación de la información, dirección de carga, longitud, etc. Este conjunto de datos se denomina CABECERA, o *HEADER* (pronunciado *híder*).

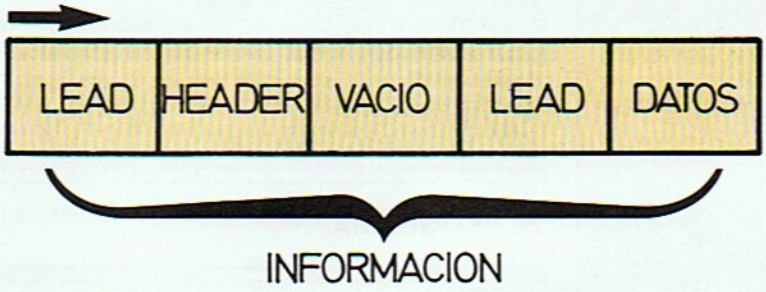
Cuando al Spectrum le ordenamos grabar una CABECERA, emite un primer tono de unos cinco segundos de duración, denominado *LEADER* (pronunciado *líder*) o simplemente *LEAD* (*«lid»*), cuyo objetivo es, en la posterior lectura, anunciar al ordenador la llegada de una serie de datos, con el fin de que éste no se vea sorprendido por el próximo alubión de bits. Por otra parte, cuando se efectúa la grabación, el *lead* generado por el Spectrum sirve a su vez para regular el volumen de grabación de aquellos casetes que efectúan esta operación automáticamente.

Tras el *lead,* llega la cabecera, portando una información que estudiaremos unas líneas más abajo de una forma más detallada. El *header* es examinado por el Spectrum, y en caso de admitirse la información a introducir, se continúa la lectura. De no ser así, o bien se produce una detención por error, o bien se ignora el bloque de datos siguiente.

Las detenciones por error pueden ser de muy diverso tipo, y van desde las muy generales, como por ejemplo el *2 Variable not found* (2 Variable no encontrada), el cual aparece al intentar grabar una matriz inexistente, hasta los muy específicos, como es el caso de *F Invalid file name* (F Nombre no válido para la grabación), el cual hace acto de presencia cuando intentamos grabar un programa sin nombre, o con una denominación cuya longitud excede los diez caracteres máximos permitidos.

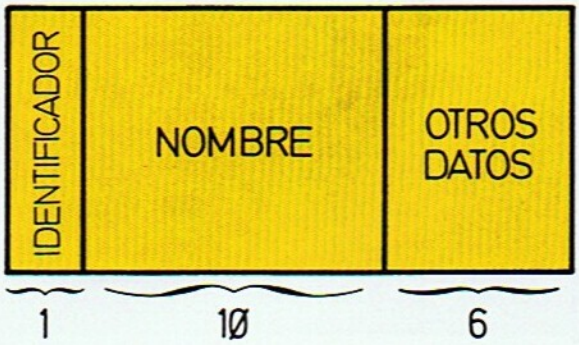
Seguido de la cabecera, y tras un pequeño intervalo de un segundo, se emite un nuevo *lead,* igual al primero y con la misma finalidad que éste, para inmediatamente después pasar a transferirse el bloque principal de información. Todo este proceso es realizado por una serie de rutinas implementadas en el FIRMWARE de nuestro Spectrum (ROM), conocidas bajo la denominación general de *CASSETTE HANDLING ROUTINES* (rutinas de manejo del casete). Esta zona de la ROM incluye el tratamiento de todos los comandos concerciente al casete (**SAVE, VERIFY, LOAD** y **MERGE**), así como las posibles modalidades de ellos (**CODE, SCREEN$**, etc...), desde su sintaxis hasta su ejecución completa.

El espacio de un segundo que el Spectrum se toma entre la cabecera y el bloque principal de información, le obliga a incluir un segundo *lead,* con duración distinta al primero por motivos que a continuación veremos. Este tiempo intermedio entre tono y tono, es aprovechado por el Spectrum, en las operaciones de lectura, para exami-



La estructura de la información almacenada en cinta por el Spectrum, se detalla en el gráfico superior.

Los diecisiete bytes de un header, se distribuyen en: identificador, nombre y datos diversos.



El programa carece de gráficos definidos, debido a lo cual no existen caracteres subrayados en el listado del mismo.

*

Para la grabación del programa introduciremos el siguiente comando: **SAVE "INDEX" LINE 10.**

*

El bloque inicial de datos que envía el Spectrum a la hora de almacenar información, se denomina CABECERA.

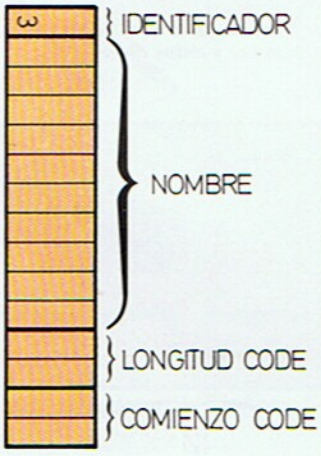
*

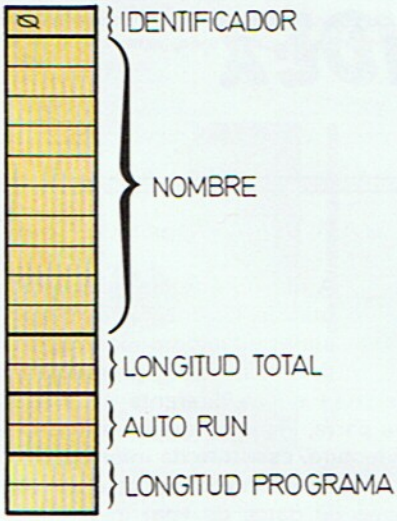
Toda la información almacenada en la ROM recibe el nombre de FIRMWARE.

nar la información aportada por el *header* y operar en consecuencia.
Así, por ejemplo, averigua si existe suficiente memoria para la entrada de la información, si el nombre de los datos coincide con los que se desean leer, o si el tipo es el adecuado (programa, bytes, matrices)... En este sentido, cuando se decide que el bloque siguiente no es el indicado, se desecha su lectura, quedando el ordenador «a la escucha» de un nuevo *lead* de cabecera; de ahí que ambos tonos, los anteriores a cabeceras y a datos, sean diferentes, con el fin de permitir al Spectrum distinguir entre la llegada de uno u otro tipo de información.

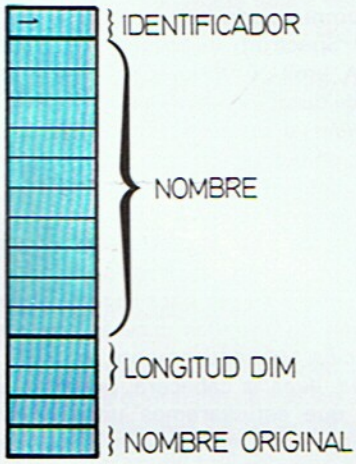
EL PROGRAMA

Nuestro programa INDEX, nos permite conocer toda la información contenida en una cabecera, así como el índice o directorio de una cinta completa. Esta segunda misión del programa es muy útil, fundamentalmente para todos los que tengamos una impresora a nuestro servicio. Gracias a él, podremos dejar funcionando el casete con la cinta de la que deseamos conocer el contenido, y conforme vayan apareciendo las cabeceras, irán siendo escritas en la impresora.
Al comienzo del programa, el Spectrum nos preguntará si deseamos la salida de datos por la pantalla o por la impresora. A continuación, si hemos escogido la segunda opción, se nos interrogará sobre el nombre de la cinta (identificador), con el fin de ser escrito al comienzo del índice.

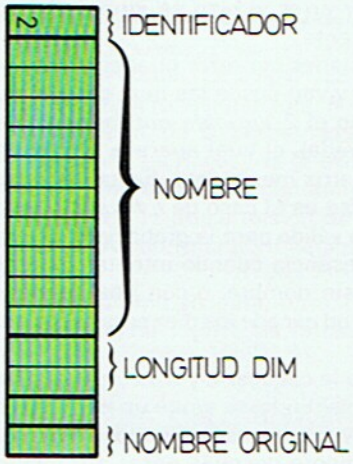




Header de una información del tipo programa.



Header de una información del tipo matriz numérica.



Header de una información del tipo matriz de cadena.

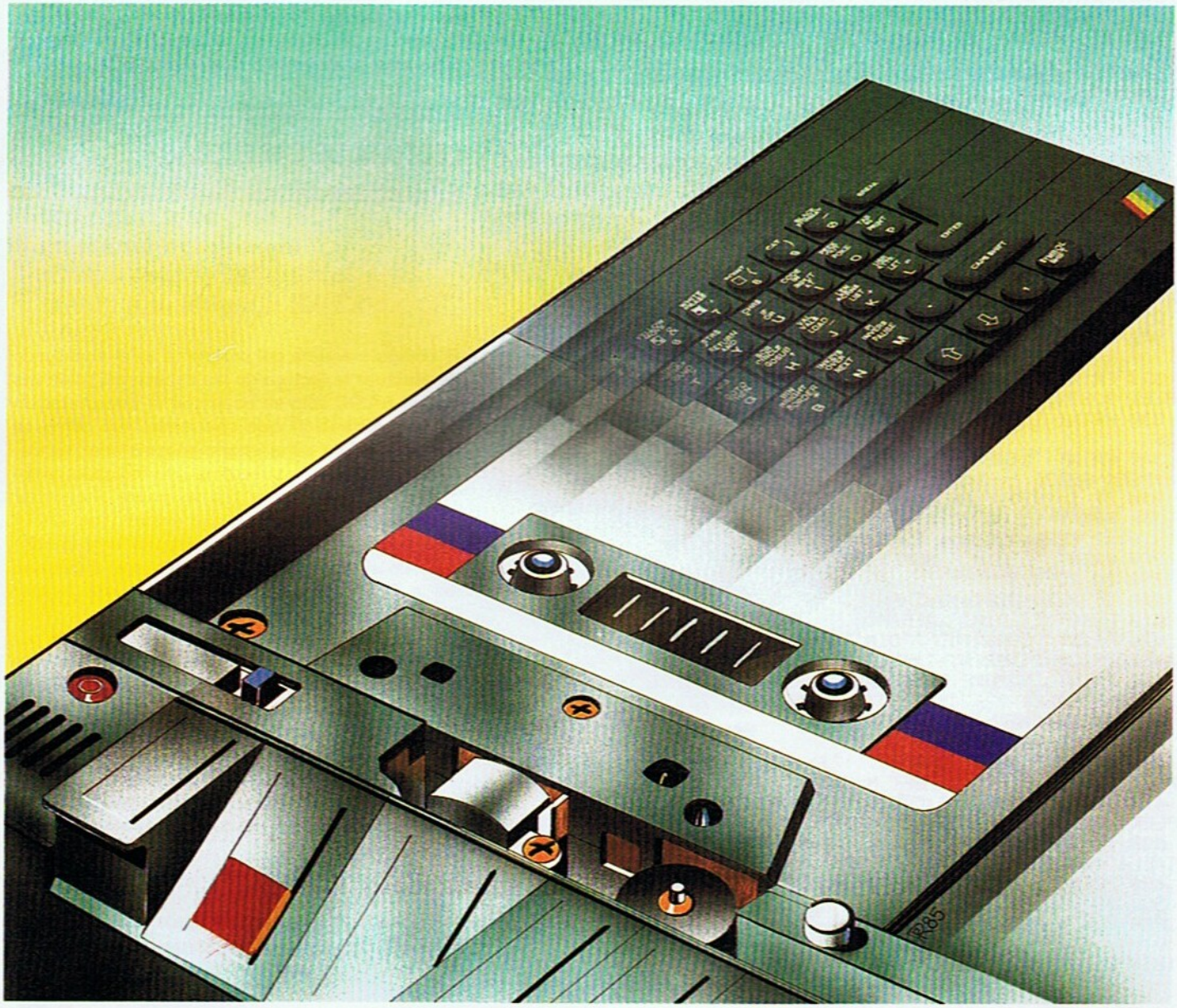
Header de una información del tipo bytes.

LISTADO ENSAMBLADOR

HEADER	175	XOR A	;Pone a cero el acumulador.
	55	SCF	;Conecta la bandera de ;acarreo.
	221 33 10 125	LD IX,$32010	;Sitúa en 32010 la dirección ;de carga de la cabecera.
	205 86 5	CALL $1366	;Llama a la subrutina ROM de ;lectura de cabeceras.
END	201	RET	;Retorna al BASIC.

En el caso de que elijamos la salida por pantalla, el Spectrum nos preguntará si deseamos que el mensaje *scroll* detenga la impresión cada vez que se llene la pantalla.

Una vez cumplidos estos requisitos previos, sólo resta poner en marcha el casete con la cinta a examinar, y por supuesto conectar los cables EAR del Spectrum, como si de una lectura cualquiera se tratase. El programa esperará pacientemente las cabeceras de la cinta, y nos brindará por el



```
   5 REM ************************
  10 REM * J.M.MAYORAL SERRANO *
  15 REM ************************
  16 REM *     INDEX  1985      *
  17 REM ************************
  20 CLEAR 31999
  30 FOR I=32000 TO 32009
  40 READ J: POKE I,J
  50 NEXT I
  60 DATA 175,55,221,33,10,125,205,86,5,201
  70 POKE 23658,8: LET S=0: LET M=0
  80 BORDER 1: PAPER 1: OVER 0: INK 7: CLS
  90 GO SUB 1000
 100 PRINT AT 2,0;"DESEA SALIDA POR IMPRESORA?"
 110 PRINT AT 2,29; OVER 1; INVERSE 1;" "
 120 IF INKEY$="S" THEN  PRINT AT 2,29;"SI": LET P=1:
 GO TO 210
 130 IF INKEY$="N" THEN  PRINT AT 2,29;"NO": LET P=0:
 GO TO 150
 140 GO TO 110
 150 IF INKEY$<>"" THEN  GO TO 150
 160 PRINT AT 4,0;"CON MENSAJE DE ~SCROLL~?"
 170 PRINT AT 4,29; OVER 1; INVERSE 1;" "
 180 IF INKEY$="S" THEN  PRINT AT 4,29;"SI": GO TO 24
0
 190 IF INKEY$="N" THEN  PRINT AT 4,29;"NO": LET S=1:
 GO TO 240
 200 GO TO 170
 210 INPUT "IDENTIFICADOR DE CINTA?";N$
 220 OPEN #2,"P": PRINT N$: PRINT
 230 GO TO 250
 240 GO SUB 1000: PRINT
 250 POKE 32010,255
 260 RANDOMIZE USR 32000
 270 IF PEEK 32010>3 THEN  GO TO 260
 280 PRINT "NOMBRE.........";: FOR I=32011 TO 32020:
PRINT CHR$ PEEK I;: NEXT I
 290 PRINT : PRINT "TIPO..........."
 300 IF PEEK 32010 THEN  GO TO 370
 310 PRINT "PROGRAMA"
 320 PRINT "LONGITUD TOTAL.:";: LET I=PEEK 32021+256*
PEEK 32022: PRINT I: LET M=M+I
 330 PRINT "LONGITUD PROG..:";PEEK 32025+256*PEEK 320
26
 340 PRINT "LINEA ""AUTORUN"":";: LET I=PEEK 32023+25
6*PEEK 32024
 350 IF I=9999 THEN  PRINT "NO HAY": GO TO 370
 360 PRINT I: GO TO 490
 370 IF PEEK 32010>1 THEN  GO TO 400
 380 PRINT """DIM"" NUMERICO"
 390 GO TO 420
 400 IF PEEK 32010>2 THEN  GO TO 450
 410 PRINT """DIM"" ALFANUM."
 420 PRINT "LONGITUD ""DIM"".:",: LET I=PEEK 32021*25
6*PEEK 32022: PRINT I: LET M=M+I
 430 PRINT "NOMBRE ORIGINAL:";CHR$ (64+32*(PEEK 32024
/32-INT (PEEK 32024/32)));"$" AND PEEK 32010=2
 440 GO TO 490
 450 PRINT "BYTES"
 460 PRINT "LONGITUD CODE..:";: LET I=PEEK 32021+256*
PEEK 32022: PRINT I: LET M=M+I
 470 PRINT "COMIENZO CODE..:";: LET J=PEEK 32023+256*
PEEK 32024: PRINT J
 480 IF I=6912 AND J=16384 THEN  PRINT "IMAGEN DE PAN
TALLA"
 490 PRINT : PRINT : IF S THEN  POKE 23692,255
 500 GO TO 250
1000 CLS : PRINT  INVERSE 1;"            I N D E X
1010 RETURN
```

medio escogido toda la información de interés: Si se trata de un programa, indicará su nombre, longitud total (programa más variables), longitud del área de variables, y línea de autoejecución (si ésta existiera).

En la lectura de bytes, anunciará su denominación, longitud y dirección de carga, así como la particularidad de si se trata de una imagen de pantalla (**SCREEN$**).

Por último, ante la lectura de matrices, nos indicará el tipo de que se trata (numérica o de cadena), su nombre, ocupación en bytes, y nombre que tenía en el área de variables en el momento de la grabación.

Además de todo esto, el programa inicializa una variable al comenzar el índice, en la cual va acumulando las longitudes de todos las informaciones examinadas; de esta manera, al llegar al final de la cinta, podemos conocer el total de bytes de información neta contenidos en ella (sin contar con los diecisiete de cada cabecera), sólo con preguntar el contenido de la variable **M**.

LECTURA DE CABECERAS

El código máquina del programa es el encargado de efectuar la lectura de las cabeceras, mientras que el BASIC sólo se ocupa de la interpretación de la información que contienen. Tal como se puede observar en el listado ensamblador, se trata de una pequeña subrutina que indica la dirección de memoria en la que se quiere almacenar la cabecera, para su posterior examen (de 32011 en adelante), y prepara el acumulador y la bandera de acarreo, para la llamada a la subrutina de la ROM que efectúa la lectura del *header* (1366 decimal).

La información de cabecera está compuesta por diecisiete bytes, de los cuales los once primeros siempre tienen una misma misión: el primero de ellos es un identificador del tipo de información recogida (0=programa, 1=matriz numérica, 2=matriz alfanumérica y 3=bytes), y los diez siguientes el nombre con que fue almacenada la información, completada con espacios en blanco por la derecha (código 32), en caso de que no llegue a los diez caracteres.

Los últimos seis bytes de información tienen un significado variable, según el identificador de datos. En todo caso, son los datos indispensables para el correcto funcionamiento de todos los comandos de casete: **SAVE, VERIFY, LOAD** y **MERGE.**

MINI PROGRAMAS

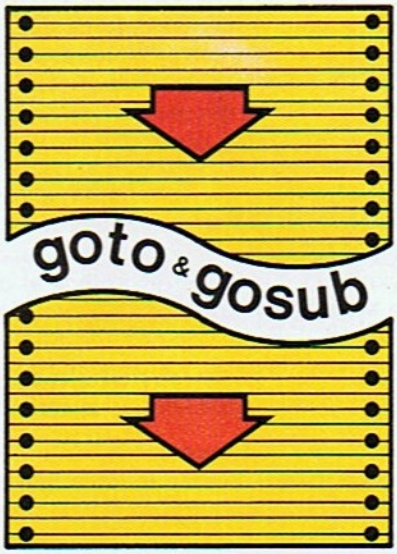
HEMOS visto hasta ahora que, si nosotros no indicamos lo contrario, el curso normal de ejecución del programa es la interpretación secuencial, una a una, de todas y cada una de las instrucciones que lo componen, hasta alcanzar la última.

La única forma que conocemos de impedir que esto suceda así, es incluir en alguna parte del programa una sentencia **GO TO**. Como sabemos, **GO TO** permite dirigir la ejecución del programa, tanto hacia adelante como hacia atrás, justo al comienzo del número de línea que indiquemos como parámetro de la sentencia.

Esta facultad nos permite saltarnos determinado bloque de instrucciones, o ejecutar un conjunto de ellos tantas veces como deseemos (bucle), en base a determinada condición, o de forma imperativa.

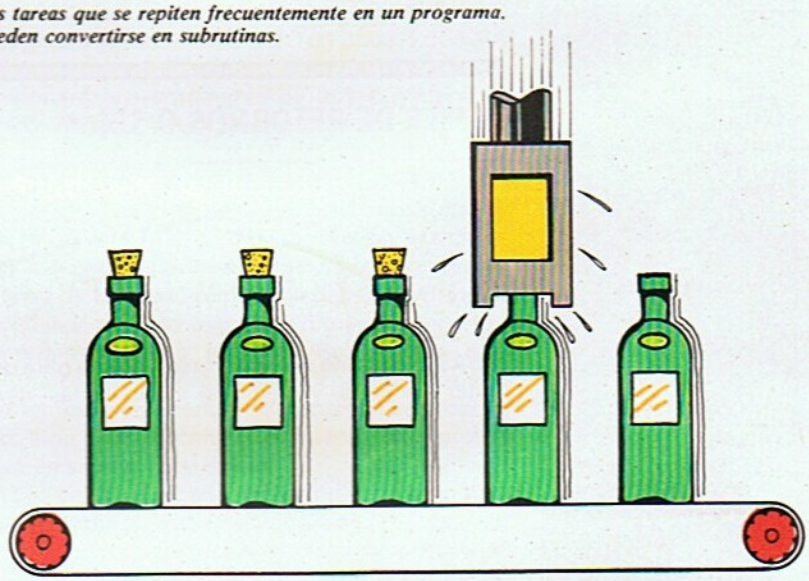
En cualquier caso, la sentencia **GO TO** no nos permite desviar la ejecución normal del programa a un bloque concreto de instrucciones, y conseguir que el ordenador «recuerde» desde qué punto del programa se le envió, para al concluir la ejecución de este «mini programa», volver al punto de partida y continuar con el programa principal.

Tanto el GO TO *como el* GO SUB *interrumpen la secuencia normal de programa.*



Para resolver situaciones de este tipo, el BASIC cuenta con la sentencia **GO SUB** (*GO SUBroutine*) que, como **GO TO**, va seguida del número de línea a saltar, pero con la condición de recordar desde qué punto exacto del programa se le envió, de forma que al finalizar la ejecución del conjunto de instrucciones deseado, retorne al punto de partida.

Las tareas que se repiten frecuentemente en un programa, pueden convertirse en subrutinas.



A este conjunto de instrucciones que ha de ejecutarse desde un punto concreto del programa, y con vuelta posterior a su origen, se le denomina SUBRUTINA.

No existe limitación en cuanto al lugar donde colocar las subrutinas de un programa BASIC, pero sí debe especificarse correctamente su principio y final. Es decir, las subrutinas pueden encontrarse al principio, al final, o en la mitad de un programa, e incluso desperdigadas sin orden ni concierto aparente, siempre y cuando sepamos su punto de partida y de final.

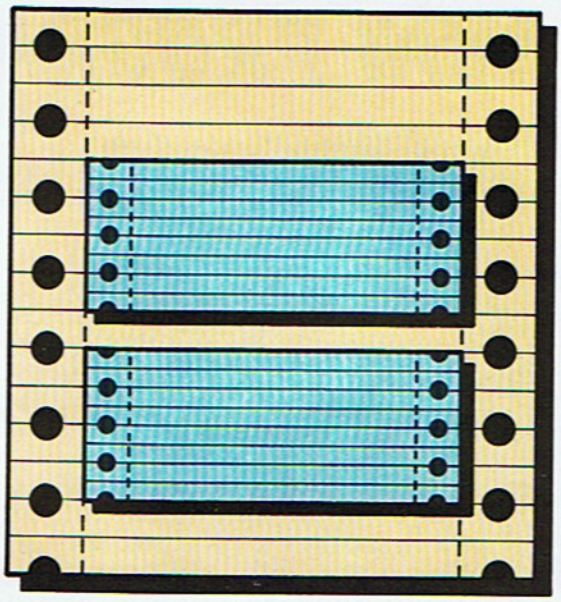
Una subrutina debe comenzar obligatoriamente al principio de una línea de programa, lo mismo que sucede con **GO TO**, puesto que el BASIC no es capaz de bifurcar a una parte concreta de una línea de instrucción. De ser así, la referencia de punto de destino habría de ser más completa, al tener que diferenciar dentro de una misma línea

las diversas instrucciones separadas por dos puntos (:).

Para identificar el final debemos incluir, en la última línea del bloque, una sentencia **RETURN** (en inglés, vuelve). Esta sentencia abandona la subrutina, y regresa exactamente a la siguiente instrucción a aquella en que se ejecutó el **GO SUB** (en inglés, vete a la subrutina) que la envió.

A pesar de que la colocación dentro del programa queda a discreción del programador, hay una regla de oro que debe respetarse, y es la obligación de colocar la subrutina en un lugar en el cual sólo sea posible el acceso a través de las instrucciones **GO SUB** del programa principal.

Debe evitarse a toda costa que el programa pueda entrar en una subrutina por otro medio diferente de **GO SUB**, como podría ser un **GO TO**, o una entrada de la propia secuencia normal del programa en la subrutina de forma incontrolada. Caso de producirse este hecho, obtendríamos el mensaje de error **RETURN without GO SUB** (RETURN sin GO SUB).



Las subrutinas se pueden considerar como «mini-programas» dentro de otro principal.

¡!

Cuando un programa entra de forma incontrolada en el área de subrutinas, se obtiene el mensaje de error **RETURN without GO SUB** (RETURN sin GO SUB).

*

El BASIC dispone de un curioso sistema para gestionar las «idas» y «venidas» a subrutinas, denominado pila de retornos o *stack*.

*

El aislamiento de las subrutinas evita que el programa pueda entrar en ellas por otro medio diferente de **GO SUB**, como podría ser un **GO TO** o una entrada de la propia secuencia normal del programa en la subrutina.

LA PILA DE RETORNOS O *STACK*

El BASIC dispone de un curioso sistema para gestionar las «idas» y «venidas» a subrutinas, denominado pila de retornos o *stack*. Para entender el funcionamiento del *stack* de forma sencilla, supongamos que nosotros mismos somos el ordenador y estamos ejecutando las instrucciones de un programa, una tras otra.

De improviso nos encontramos con una sentencia **GO SUB**, que nos ordena saltar a determinado número de línea, ¿qué hacer? ¡Está claro!, anotamos en un papel el número de instrucción por el que vamos, lo dejamos encima de la mesa, y pasamos a ejecutar la instrucción que se nos ordena, como si de un **GO TO** convencional se tratara.

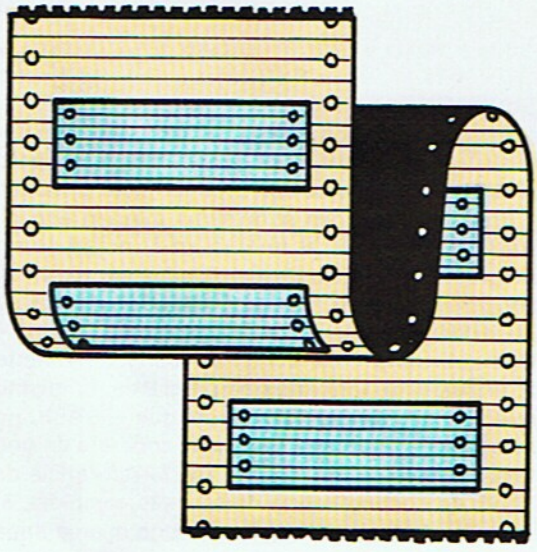
Imaginemos ahora, complicando un poco las cosas, que encontramos otra sentencia **GO SUB** dentro de la subrutina. Lo evidente es que, del mismo modo que antes interrumpimos el programa principal para ejecutar una subrutina, debemos actuar ahora en consecuencia.

De esta forma, nos vemos dentro de una estructura anidada, similar a las descritas al hablar de los bucles de programación **FOR-NEXT**, o lo que es lo mismo, inmersos en una subrutina de subrutina.

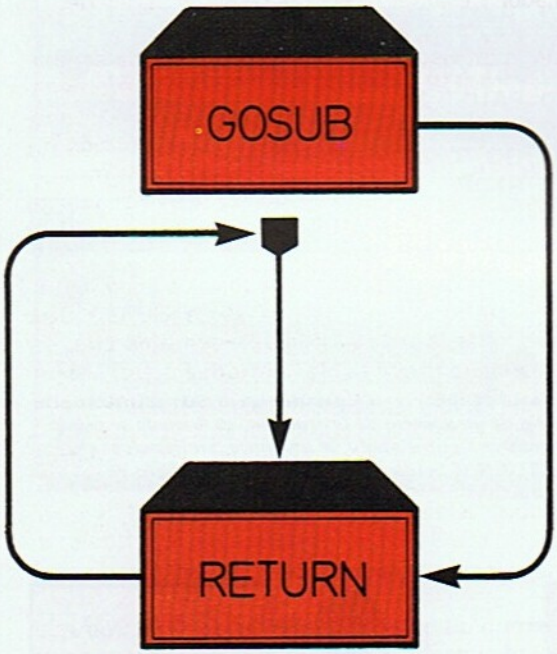
Para solucionar el problema, recurrimos al procedimiento antes descrito, anotando en otro papel el número de línea en que nos encontramos, y colocándolo sobre la mesa, justo encima del anterior. Hecho esto, podemos efectuar el salto que se nos indica.

Supongamos ahora que tropezamos con un **RETURN**, ¿dónde volver? Pues a la línea indicada en el último papel colocado sobre la mesa, dado que corresponde forzosamente a la última subru-

No existe limitación en cuanto al lugar en que se sitúan las subrutinas dentro de un programa.

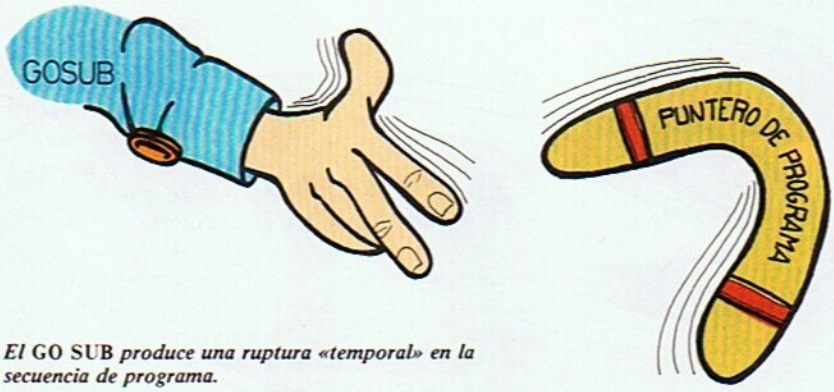


tina en ejecución, por lo cual retiramos el primer papel visible, y continuamos con la ejecución del programa desde el punto ya mencionado.
Siguiendo la ejecución del programa, más tarde o más temprano, encontraremos otro **RETURN** y, esta vez, tomaremos el último papel que queda sobre la mesa, retornando al programa principal.
En base al comportamiento observado, podemos sacar fácilmente la conclusión de que, en el *stack,* el último en entrar es el primero en salir.
Este sistema, que de aplicarse en las colas de los cines provocaría auténticas batallas campales, es el idóneo, sin embargo, para el manejo de subrutinas, por permitir su ejecución anidada de forma correcta, y similar a como sucede cuando se anidan bucles del tipo **FOR-NEXT**.
Al tipo de almacenamiento de direcciones de retorno en el *stack,* se le denomina *LIFO (Last In First Out),* en contraposición con el método normal de ejecutar los trabajos que es el *FIFO (First In First Out).*
El manejo de esta pila de retornos no nos debe preocupar en absoluto, dado que es gestionada automáticamente por el Sistema. No obstante, debemos saber que el mantener un gran número de subrutinas «abiertas», es decir sin **RETURN,** puede ocasionarnos, en determinado momento, un problema de desbordamiento del *stack,* produciendo el error **4 Out of memory** (fuera de memoria).



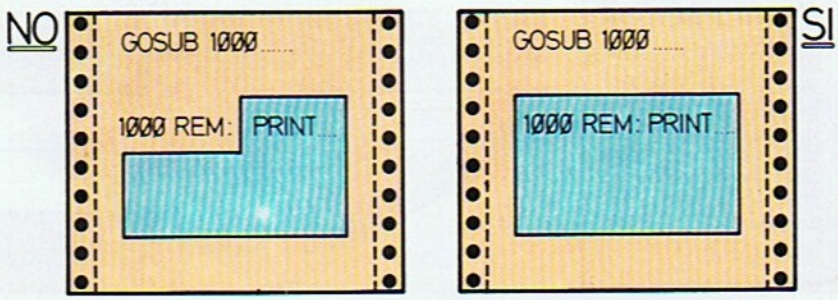
RETURN *produce el retorno al punto inmediatamente posterior al de la última llamada.*

En la práctica es casi imposible que llegue a darse este problema, puesto que el *stack* del Spectrum es dinámico, y puede crecer desde el final de la memoria RAM hasta su comienzo, tanto como sea necesario.
A pesar de ello, se debe vigilar que la salida de las subrutinas se produzca siempre con **RETURN,** y no con una sentencia **GO TO**, puesto que **GO TO** no restaura el *stack* y, aunque el programa funcionará aparentemente de forma normal, no constituye una técnica adecuada de programación.
Así pues, para tener una apreciación algo más precisa del funcionamiento concreto del *stack* del Spectrum, debemos imaginar que en vez de situar los papeles con las direcciones de retorno encima de la mesa, utilizamos papeles adhesivos los cuales, una vez escrito el número a recordar, adherimos al techo, en forma de columna, pero no una columna normal, en la cual cuantos más elementos la componen más próxima está del techo, sino todo lo contrario: cuanto más grande es, más se acerca al suelo.
Esto se debe a que la información correspondiente a programas, variables, etc... se encuentra al principio de la memoria (digamos que en el «suelo»); si el *stack* se situara a continuación de és-



El **GO SUB** *produce una ruptura «temporal» en la secuencia de programa.*

Las subrutinas siempre deben comenzar al inicio de una línea de instrucción.



El RETURN *produce el retorno desde la última subrutina llamada.*

tos, tendría que desplazarse cada vez que hubiera modificaciones en la longitud del programa, el área de variables, etc... Por el contrario, siguiendo el sistema de *stack* «adherido al techo», ambas zonas de memoria pueden funcionar de manera independiente.

Otra posible solución al problema del *stack*, sería establecer una zona determinada de la memoria con destino a él, situada en un lugar fijo, por ejemplo, después de la pantalla y antes del programa. Dado que este área no podría ser utilizada para ningún otro fin, no podríamos hacerla demasiado grande, pues supondría un desperdicio de memoria. De igual modo, si la zona destinada al *stack* es excesivamente pequeña restringiremos el número de subrutinas anidadas.

Este último sistema de *stack* es el que adoptan algunos ordenadores, que generalmente pueden tener un máximo de 25 niveles de anidamiento, es decir, 25 llamadas a subrutinas pendientes de retorno. Por el contrario, y a modo indicativo de la eficacia del *stack* dinámico del Spectrum, diremos que en el modelo de 16 K se pueden llegar a alcanzar más de 2.800 anidamientos, y en el de 48 K o PLUS (que en capacidad de memoria son idénticos) ¡más de 13.700!

¡!

Se dice que una estructura de subrutinas es anidada, cuando unas se encuentran completamente dentro de otras, de forma similar al anidamiento descrito para los bucles del tipo **FOR NEXT**.

*

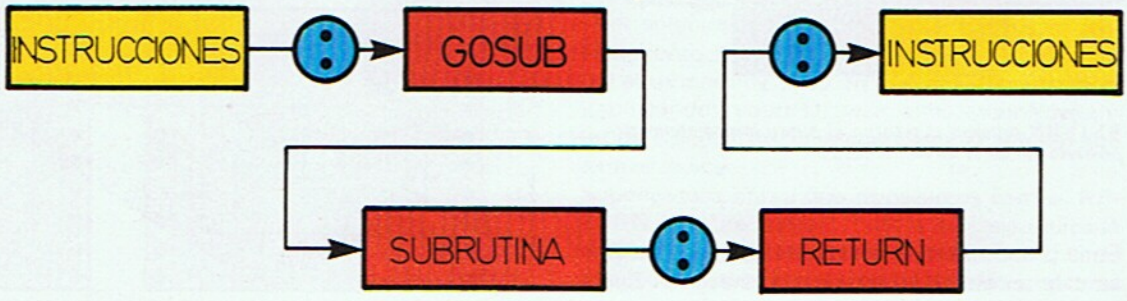
Se denomina «recursividad» a la técnica de programación que se emplea cuando un mismo bloque de programa se llama a sí mismo, cuantas veces sea necesario, hasta resolver el problema concreto que se le encomienda.

ALGUNOS EJEMPLOS PRACTICOS

Dos son las ventajas principales que ofrece el empleo de subrutinas dentro de nuestros programas. La primera de ellas se refiere a la calidad del mismo, puesto que un programa bien estructurado debe estar codificado de forma modular, es decir, por bloques de instrucciones que ejecuten una labor determinada, y esto se manifiesta muy de acuerdo con la definición de subrutina. La segunda de las ventajas es cuestión de cantidad. Si una misma tarea se repite en diversos puntos de un programa, no sería muy inteligente por nuestra parte escribir las mismas instrucciones todas las veces. El método a seguir en estos casos es programar una subrutina con el bloque de líneas que se repiten, y luego acceder a él siempre que sea necesario mediante **GO SUB**. Como ejemplo de lo comentado vamos a ver las evoluciones de un platillo volador, apoyándonos en el empleo de subrutinas:

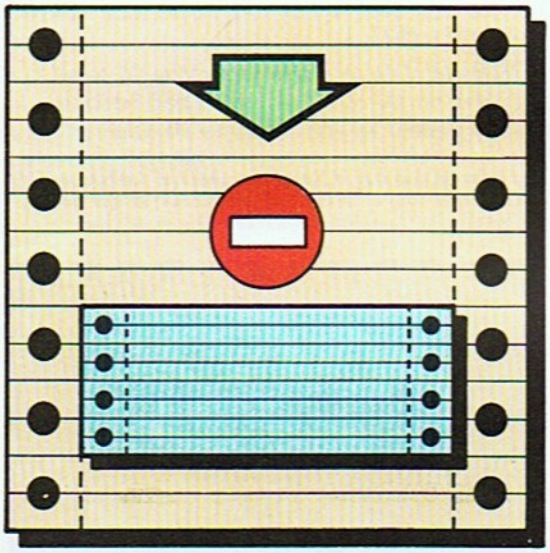
```
10  REM - SUBRUTINAS
20 LET X=RND*20: LET Y=RND*28
30 LET X$="123333": GOSUB 80
40 PAUSE 10
50 LET X$="        ": GOSUB 80
60 GO TO 20
70 STOP
80 PRINT AT X,Y+1;X$( TO 2); AT X+1,Y;
X$(3 TO ): RETURN
```

Aunque el comienzo de una subrutina tiene que estar al inicio de un número de instrucción, su llamada se puede encontrar en otro punto de una línea, realizándose el RETURN *al punto posterior en la misma línea de instrucción.*

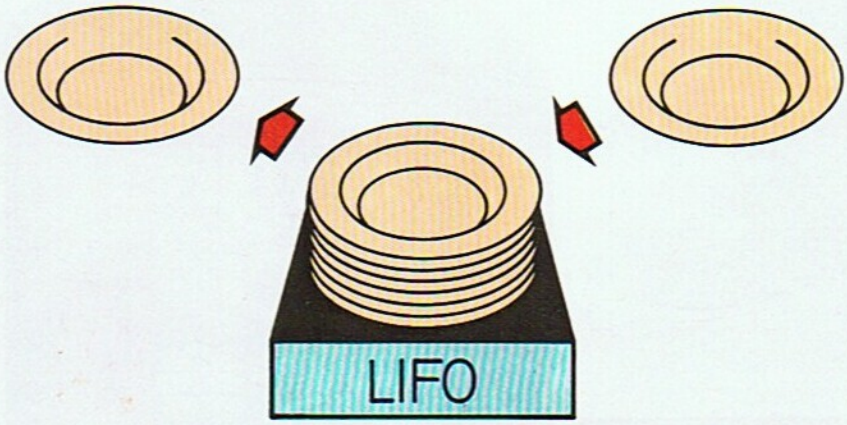


En la línea 20 se definen los valores de la abscisa y ordenada para la impresión del platillo. Como podemos ver, en la generación aleatoria se controla que el platillo no pueda caer fuera de los límites de la pantalla.
En la línea 30 se asigna valor a la variable **X$** que contiene la serie de caracteres a imprimir, a través de la subrutina de la línea 80.
La línea 40 tiene el cometido de establecer una pausa, para poder observar las evoluciones del platillo.
La línea 50 es de contenido similar a la 30, pero esta vez contiene espacios en lugar de los caracteres gráficos que componen el platillo. Con esto logramos, aprovechando una subrutina común, acceder tanto a la escritura como al borrado del móvil.
La línea 60 cierra el ciclo de programa con un **GO TO** imperativo a su comienzo.
La línea 70 tiene una misión fundamental. Como ya dijimos antes, es necesario proteger la zona de subrutinas del programa, para que sólo pueda llegarse hasta ella por medio de las sentencias **GO SUB**.

No debemos permitir que el curso del programa se introduzca en el área de subrutinas por un camino distinto al GO SUB.

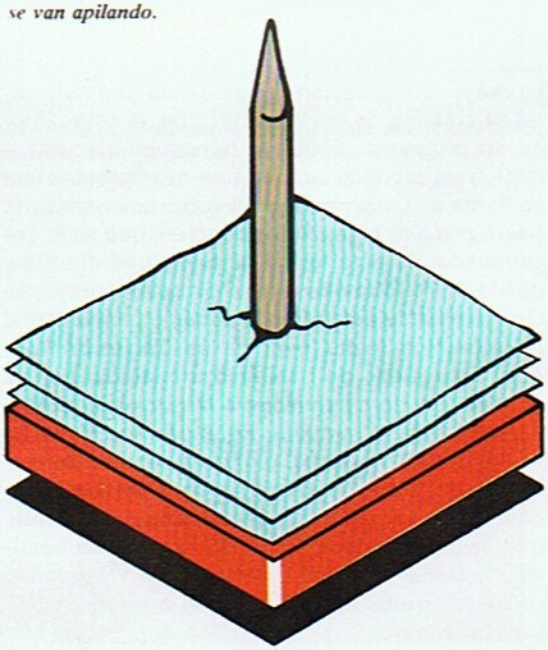


El **STOP** de la línea 70 es, por esta vez, un mero adorno, puesto que el programa nunca pasa por él, evitándose la entrada incontrolada en el **GO TO** de la línea anterior. No obstante, lo incluimos para servir como norma de la separación que debe existir siempre entre las líneas del programa principal y el área de subrutinas.
Por último, en la línea 80 se gestiona la impresión en la pantalla, de acuerdo con las coordenadas **X** e **Y**, del valor de la variable **X$** que, unas veces contiene los gráficos del platillo, cumpliendo la función de escritura, y otras espacios, implicando el borrado de la impresión anterior.
El ejemplo propuesto es el más simple de utilización de subrutinas, dado que en el caso más desfavorable, permanece sólo una dirección de retorno en el *stack* del Sistema. Esto es lo mismo que decir que no se produce ningún anidamiento, o sea, una llamada a subrutina desde dentro de otra subrutina.



En el sistema L.I.F.O., el último elemento en entrar es el primero en salir.

En el stack, las anotaciones con las direcciones de retorno se van apilando.

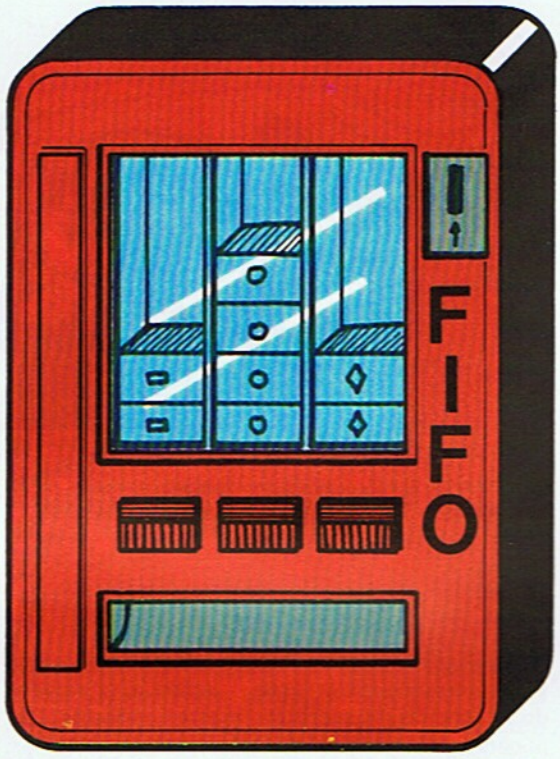


¡!

Definir un determinado bloque de programa como subrutina, revierte en un importante ahorro de espacio en la memoria, pudiendo accederse a él desde cualquier punto del programa.

*

Un programa bien estructurado debe estar codificado de forma modular, es decir, por bloques de instrucciones que ejecuten una labor determinada, y esto se manifiesta muy de acuerdo con la definición de subrutina.



En el sistema F.I.F.O., el primer elemento en entrar es el primero en salir.

Los movimientos se producen peldaño a peldaño, y teniendo cuidado de colocar cada uno de ellos sobre otro de mayor longitud. Lógicamente, como si de una construcción se tratase, nunca puede haber un peldaño mayor sobre otro menor. Siguiendo estas reglas, las piezas se pueden ir pasando a cualquiera de los tres bastidores, hasta conseguir formar la torre inicial en uno de los dos vacíos al comienzo.

La última norma del juego, ilustra claramente el sistema de *stack* L.I.F.O., puesto que sólo podemos mover aquellas piezas que se encuentren sobre cualquiera de las torres; es decir, si queremos desplazar la primera de las piezas, habremos de utilizar primero los movimientos necesarios para apartar las que tenga encima.

Naturalmente, hemos de procurar resolver el problema en el menor número de jugadas posibles, ¡y en eso el Spectrum es un jugador eficacísimo! Cualquiera que sea el número de piezas de la torre, la cantidad óptima de movimientos seguirá la fórmula: **2↑A-1** (dos elevado a **A**, menos uno), donde **A** es la altura de la torre.

```
  10 REM TORRES DE HANNOI - J.M.LOPEZ MA
RTINEZ
  20 INPUT 'Altura (2-9):';A: IF A<2 OR
A>9 THEN  GO TO 20
  30 DIM T(3,A): DIM S(A,2): DIM U(3)
  40 LET S$='': FOR I=1 TO A: LET S$=STR
$ I+S$: NEXT I
```

¡!

Debe respetarse la obligación de colocar la subrutina en un lugar en el cual sólo sea posible el acceso a través de las instrucciones **GO SUB** del programa principal.

*

La sentencia BASIC **GO SUB** va seguida del número de línea a saltar, con la condición de recordar desde qué punto exacto del programa se le envía, de forma que al concluir la ejecución del conjunto de instrucciones deseado, retorne al punto de partida.

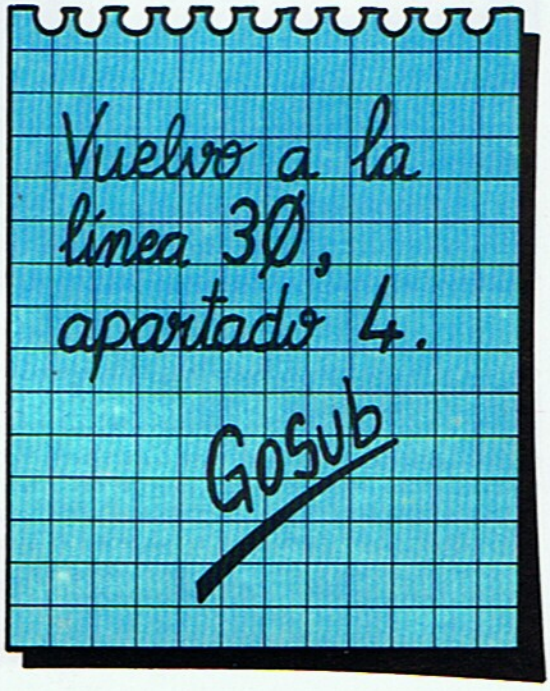
*

Una subrutina debe comenzar obligatoriamente al principio de una línea de programa, lo mismo que sucede con **GO TO**, puesto que el BASIC no es capaz de bifurcar a un segmento de instrucción de los separados por dos puntos (:).

SOFISTICACION EN LAS SUBRUTINAS

La mayor sofisticación llega cuando una subrutina se codifica de forma recurrente. A esta técnica de programación se le denomina «recursividad», y se produce cuando una misma subrutina se llama a sí misma cuantas veces sea necesario hasta resolver el problema concreto que se le encomienda; de esta forma, al cumplirse el último de los **RETURN** se retorna al programa principal. Un ejemplo de esta técnica podemos encontrarlo en la resolución del juego de las «Torres de Hannoi». Este juego de habilidad intelectual consta de tres bastidores situados a la izquierda, centro y derecha de la pantalla. En el primero de ellos, se encuentra una torre, de altura variable dependiendo del grado de dificultad de la partida, compuesta por varias piezas en tamaño decreciente, que el jugador debe situar en otro de los bastidores, conservando el mismo orden, y siguiendo unas determinadas normas que a continuación estudiaremos.

La información que el Spectrum anota al realizar un **GO SUB**, *no es sólo el número de instrucción al que debe regresar, sino también el apartado dentro de ésta.*



```
  50 PRINT TAB 6;"- TORRES DE HANNOI -"
  60 FOR I=1 TO A: LET T(1,I)=I: NEXT I
  70 FOR I=1 TO A: LET S(I,1)=1: LET S(I
,2)=I: NEXT I
  80 LET U(1)=A
  90 GO SUB 130
 100 GO SUB 140
 110 PRINT AT 12,8;"FIN DE PROGRAMA"
 120 STOP
 130 INPUT AT 11,0;"": FOR I=1 TO 3: FOR
 J=1 TO A: PRINT AT 22-J,10*(I-1);("8888
888888")( TO A-T(I,J)+1) AND T(I,J): NEX
T J: NEXT I: RETURN
 140 LET M=VAL S$(LEN S$)
 150 LET T=S(M,1): LET H=S(M,2)
 160 IF H=U(T) THEN  GO TO 190
 170 LET S$=S$+STR$ T(T,H+1)
 180 GO TO 140
 190 LET D=T+(M/2=INT (M/2))-(M/2<>INT (
M/2))
 200 IF D=0 THEN  LET D=3: GO TO 220
 210 IF D=4 THEN  LET D=1
 220 IF NOT U(D) THEN  GO TO 260
 230 IF T(D,U(D))<M THEN  GO TO 260
 240 FOR K=1 TO U(D): IF T(D,K)>M THEN
LET S$=S$+STR$ T(D,K)
 250 NEXT K: GO TO 140
 260 PRINT AT 3,6;"MUEVO DESDE ";T;" HAS
TA ";D: PAUSE 0
 270 LET T(D,U(D)+1)=M
 280 LET T(T,H)=0
 290 LET S(M,1)=D: LET S(M,2)=U(D)+1
 300 LET U(T)=U(T)-1: LET U(D)=U(D)+1
 310 IF LEN S$=1 THEN  GO SUB 130: RETUR
N
 320 LET S$=S$( TO LEN S$-1)
 330 GO SUB 130
 340 GO TO 140
```

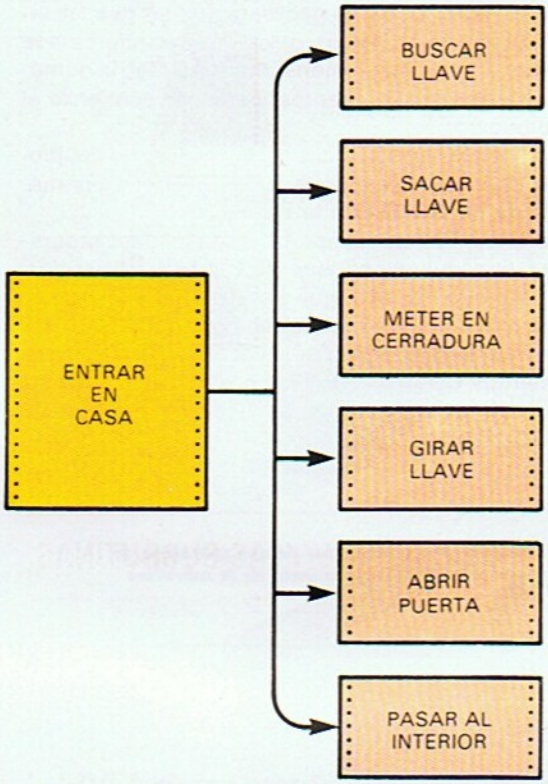
En la línea 20 se solicita la altura de la torre inicial que debe oscilar entre 2 y 9. De no encontrarse la respuesta entre los límites fijados, vuelve a pedirse el dato.

En la línea 30 se dimensionan las tablas T, S y U. Todas ellas son numéricas, y las dos primeras de dos dimensiones, siendo la tercera un vector (unidimensional). En el caso de las dos primeras, el dimensionado depende del valor de la variable **A** (número de peldaños).

La matriz **T(3,A)** almacena el contenido de cada posición dentro del bastidor. **T(2,4)** significa, por ejemplo, el contenido del cuarto peldaño, contando desde abajo hacia arriba, del bastidor central, puesto que los bastidores los hemos numerado 1, 2 y 3, de izquierda a derecha.

La matriz **S(A,2)** contiene la situación de cada peldaño. El primero de los índices señala la longitud del peldaño, que por tanto, puede oscilar entre 1 y el número de piezas que compongan la torre. Cuando el segundo índice es 1, se obtiene el número de bastidor en que se encuentra la pieza y cuando el segundo índice es 2, la altura dentro del bastidor. Así pues, si por ejemplo la pieza de longitud uno, se encuentra en el bastidor central (número 2), y en su base (altura 1), el elemento **S(1,1)**, adoptará el valor 2 (bastidor), y el **S(1,2)**, el valor 1 (altura).

En la línea 40, se efectúa la carga de la variable **S$**, la cual simula el funcionamiento de un *stack*, puesto que almacenamos de entrada en ella los movimientos que es necesario realizar.



El sistema de la programación estructurada recomienda la división de un problema en subproblemas más fácilmente tratables; éstos pueden ser representados en BASIC mediante subrutinas.

Se llaman subrutinas anidadas aquellas que son llamadas desde otras subrutinas.

¡!

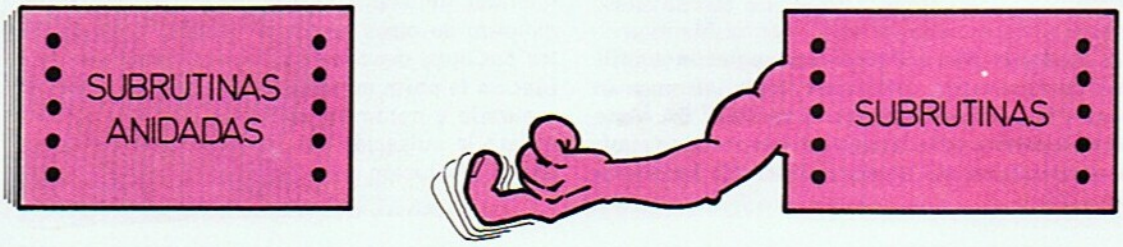
Al conjunto de instrucciones que ha de ejecutarse desde un punto concreto del programa, y con vuelta posterior a su origen, se le denomina SUBRUTINA.

*

Se debe vigilar que la salida de las subrutinas se produzca siempre con **RETURN** y no con una sentencia **GO TO**, puesto que **GO TO** no restaura el *stack* y, aunque el programa funcionará aparentemente de forma normal, no constituye una técnica adecuada de programación.

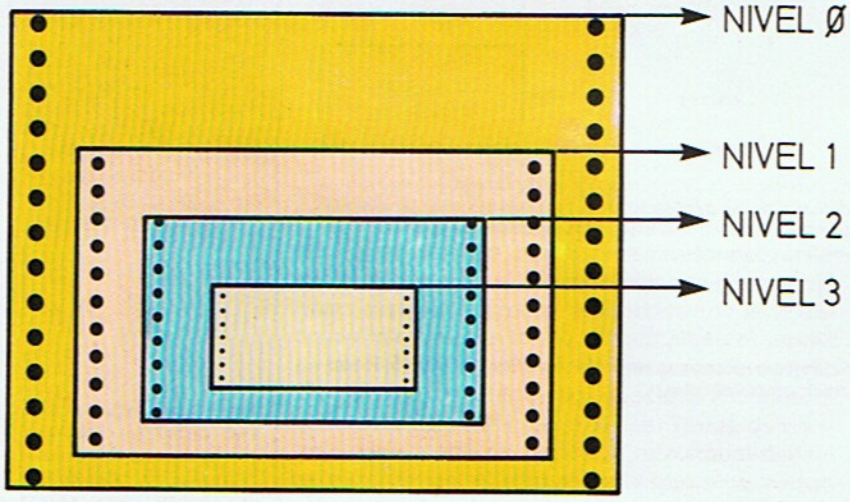
*

Para identificar el final debemos incluir, en la última línea del bloque, una sentencia **RETURN**. Esta sentencia abandona la subrutina y regresa exactamente a la siguiente instrucción que sigue a la **GO SUB** que le envió.



La línea 50 efectúa la presentación de la cabecera de pantalla, y las instrucciones 60 y 70, la inicialización de valores de las tablas **T** y **S**, respectivamente.
Para finalizar las inicializaciones, en la línea 80 se asigna la altura actual del primer bastidor (**A**), en el vector **U**. No es necesario definir que las alturas máximas de los otros dos bastidores son cero, dado que al dimensionar una matriz numérica todos sus elementos toman de comienzo el valor cero.
La línea 90 imprime la situación de inicio del juego, definida en la fase de inicializaciones, por medio de un **GO SUB** a la línea 130.
La línea 100 es realmente el núcleo del programa, puesto que contiene un **GO SUB** a la subrutina recursiva 140, que gestiona la totalidad de los movimientos necesarios para solucionar las Torres de Hannoi, en el menor número de movimientos (**2↑A-1**). De hecho, a la vuelta de esta subrutina, se imprime el mensaje de "FIN DE PROGRAMA", en la línea 110.

El número de anidamientos de una subrutina, se denomina nivel de anidamiento de la subrutina.



En las líneas 140 a 340 se encuentra la subrutina de movimiento de piezas, hasta alcanzar la solución que llega cuando la variable **S$** tiene un sólo carácter. De no ser así, la subrutina vuelve a ejecutarse por medio del **GO TO 140** de la línea 340.

El stack dinámico del Spectrum crece desde el final de la memoria hacia al principio.

La variable **M** indica el movimiento a ejecutar, tomado del extremo derecho de la variable **S$**, del mismo modo que **H** indica la altura, extrayéndola de la matríz de posiciones **S**.
En la línea 160 se evalúa si **H** es el peldaño más alto de la torre. De no ser así, se almacena en el «*stack*» **S$** el movimiento de la pieza que impide el desplazamiento deseado, recurriendo a la propia subrutina para calcular un nuevo movimiento.
En las líneas 190 a 210 se evalúa el destino **D**, cuando es posible el movimiento, en base a que el peldaño a desplazar sea o no impar. Los pares mueven siempre al bastidor de su derecha y los impares al de su izquierda, controlándose el hecho de que, un desplazamiento a la derecha en el tercer bastidor implica el movimiento al primero, del mismo modo que el desplazamiento a la izquierda, cuando nos encontramos en el primero, implica el salto al tercero.

La técnica de la recursividad consiste en el acceso reiterado a un determinado bloque desde él mismo, hasta la resolución del problema que se le ha encomendado.

De esta forma, el programa sigue filtrando movimientos, almacenando los imposibles por impedimento de otros peldaños en **S$** y efectuando los posibles, describiendo el movimiento a efectuar en la parte superior de la pantalla. Una vez realizada y notificada cada jugada, el programa espera la pulsación de una tecla para continuar con la resolución del problema.

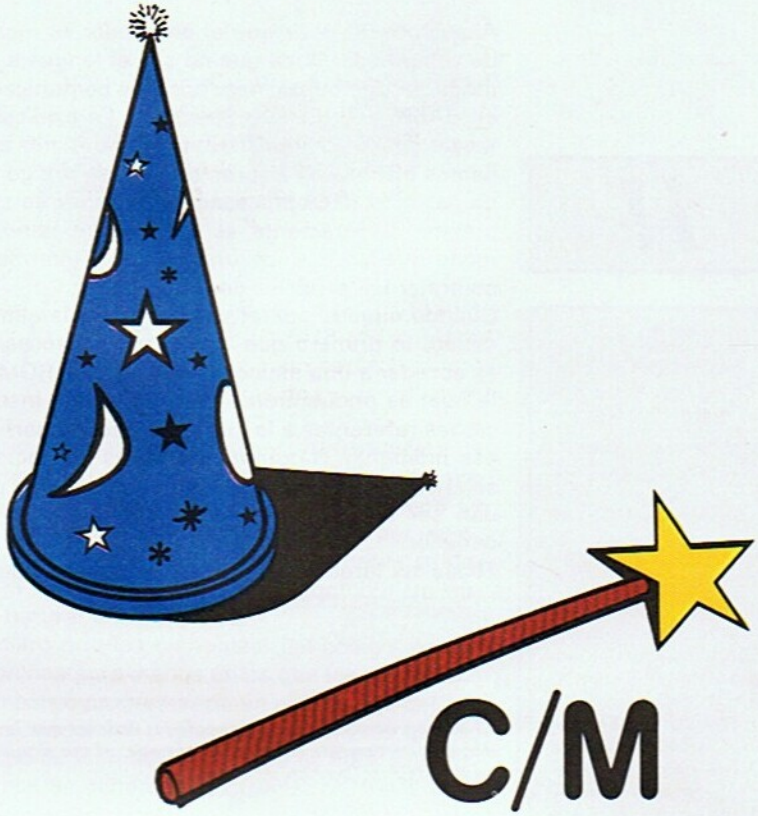
EL CODIGO MAQUINA

SIN duda alguna, para muchos usuarios del Spectrum, el código máquina es una extraña brujería, la cual les permite llevar a cabo en su aparato tareas que de otro modo serían totalmente imposibles. Esta sensación se traduce en un cierto miedo a la hora de manejarlo, y desde luego, a algunos errores de base que producen a la larga fracasos a la hora de intentar incorporar subrutinas en este lenguaje a nuestros programas. Vamos a continuación a aclarar algunas ideas sobre el tan temido código máquina.

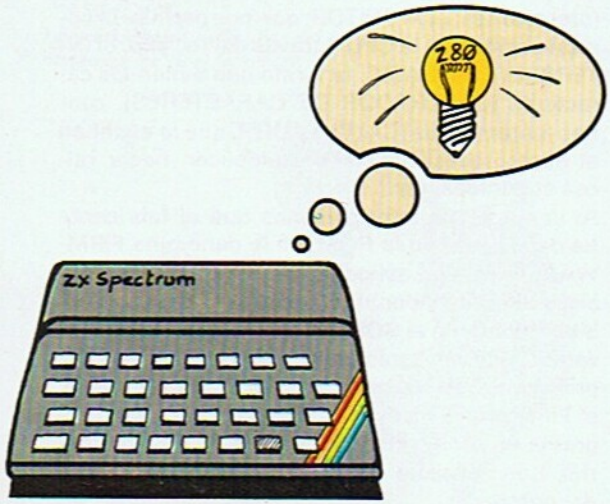
Hemos de comenzar diciendo que, cualquiera que sea el lenguaje de programación que manejemos, éste acabará siendo convertido en código máquina. Para comprender esta afirmación, tenemos que partir de una cierta base sobre el funcionamiento de nuestro ordenador.

Nuestro aparato inicialmente, cuando está desconectado, no es más que un conjunto inanimado de componentes electrónicos. Ahora bien, al fluir la chispa eléctrica por sus circuitos puede convertirse en un juguete, un profesor, o incluso un amigo ¿a qué es debido este prodigioso proceso de transformación?

El cerebro de nuestro ordenador es el microprocesador Z-80A, un pequeño *chip* capaz de efectuar por si solo las operaciones que en el albor de la informática realizaba una máquina que ocupaba toda una habitación. Al igual que los humanos entendemos un determinado lenguaje en el que se nos pueden dar órdenes o explicaciones, el microprocesador habla un lenguaje electrónico, en el cual se deben codificar las instrucciones para que él llegue a entenderlas.



Debemos desterrar la idea de que el código máquina es algo que funciona por arte de magia.



El cerebro de nuestro ordenador es el microprocesador Z-80.

Como habremos supuesto, este lenguaje está formado por impulsos eléctricos, que nosotros representamos por unos y ceros (bits), e intentar comprenderlo nos es relativamente complicado, aunque no tanto como lo sería para el ordenador comprender el nuestro; ello se debe a que el lenguaje del Z-80 es incomparablemente más limitado que el humano, y carece de imprecisiones: este lenguaje se denomina LENGUAJE MAQUINA o CODIGO MAQUINA.

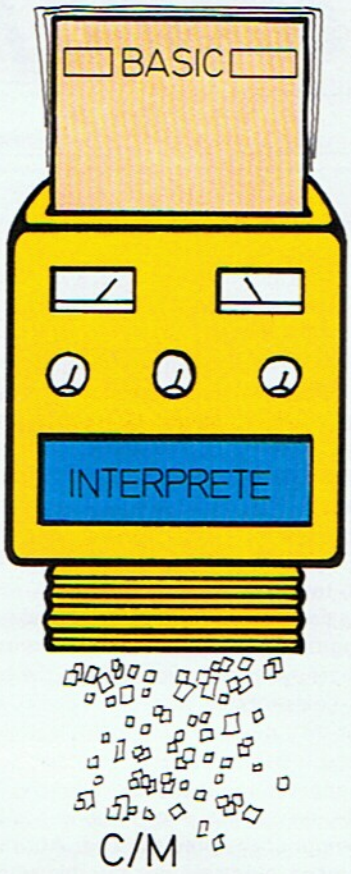
FIRMWARE

Acabamos de decir que el ordenador es incapaz de comprender nada que no sea el lenguaje máquina, sin embargo, nosotros nos comunicamos en BASIC con nuestro Spectrum. La explicación a este hecho es bien clara; ni los humanos estábamos dispuestos a aprender el lioso código máquina, ni el microprocesador era capaz de comprender directamente el lenguaje humano, de modo que adoptamos una solución intermedia: contratar los servicios de un traductor.

Cuando nuestro aparato se conecta a la alimentación, lo primero que hace el microprocesador es acceder a una memoria denominada ROM, en la cual se encuentran almacenadas las instrucciones referentes a lo que debe hacer a partir de ese momento. Naturalmente, estas indicaciones se encuentran en código máquina, para que puedan ser comprendidas directamente por el ordenador.

Todas las tareas que se efectúan al encender se encuadran en un proceso que se denomina INICIALIZACION, y van desde el borrado de la memoria RAM (del usuario), hasta la impresión del mensaje de presentación (© *1982 Sinclair Research Ltd*). Ahora bien, en la ROM no sólo se encuentra esta información, sino también otras muy interesantes; el MONITOR, que nos permite la comunicación con la CPU a través del teclado, el INTERPRETE del BASIC, la forma que tienen los caracteres (GENERADOR DE CARACTERES), rutinas matemáticas (CALCULADOR) que le enseñan al microprocesador como multiplicar, hacer raíces cuadradas, etc...

Al conjunto de informaciones que el fabricante ha depositado en la ROM, se le denomina FIRMWARE y es algo así como los conocimientos básicos del ordenador, que nosotros podemos combinar mediante el SOFTWARE en RAM, para conseguir realizar tarás más complicadas (juegos, programas de contabilidad, etc...). En definitiva, el FIRMWARE no es más que un complicado programa en código máquina (ocupa 16 K de memoria), que contiene los conocimientos de partida del ordenador.



Al fin y al cabo, programemos en el lenguaje que programemos, todo se acabará convirtiendo al código máquina.

El microprocesador no comprende las órdenes que le damos directamente en nuestro lenguaje, ni tan siquiera en BASIC.

¡!

A los circuitos integrados se les denomina familiarmente *chips*.

*

El lenguaje que maneja directamente un microprocesador se conoce como CODIGO MAQUINA o LENGUAJE MAQUINA. En muchas ocasiones encontraremos su abreviatura C/M, o la abreviatura inglesa M/C *(machine code)*.

*

El software que el fabricante incorpora en la memoria ROM de nuestro aparato, se denomina *firmware*.

*

Las instrucciones del código máquina no se expresan con letras y palabras, sino mediante números.

UN LENGUAJE DE NUMEROS

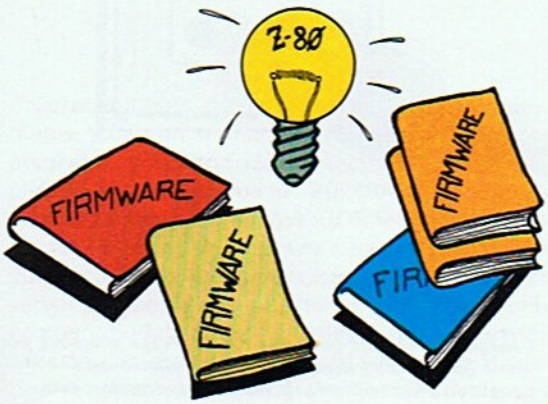
Así pues, dado lo que acabamos de ver, estamos en condiciones de comprender por qué cualquier taréa que llevemos a cabo en el ordenador, se encuentra codificada en lenguaje máquina. Así por ejemplo, cuando pulsamos una tecla, una parte del firmware se encarga de notificarle al ordenador cual de ellas hemos pulsado; de forma parecida, al introducir un programa BASIC, estamos ejecutando la rutina en código máquina, dentro de la ROM, denominada EDITOR, y cuando ejecutamos el programa BASIC que hemos introducido, ponemos en funcionamiento el INTERPRETE, para que traduzca las instrucciones a código máquina.

El lenguaje máquina no está compuesto por palabras como el BASIC, sino por números. Si le decimos al Z-80 que ejecute la instrucción 60, él entenderá que debe incrementar en uno el contenido de la variable A, es decir, algo así como si en BASIC escribiéramos LET A=A+1; si por ejemplo tuviera que ejecutar un código 201, lo entendería como un retorno de subrutina, es decir, exactamente igual que un **RETURN** en el BASIC. Como podemos ver, el código máquina no es cosa de brujería, sino simplemente un lenguaje más, con algunas similitudes con el BASIC, aunque mucho más restringido. Cuando lo estudiemos más en profundidad, tomaremos una conciencia completa de su funcionamiento, y veremos que la variable A de la cual hemos hablado en el ejemplo de unas líneas más arriba (ejecución de un código 60), no es la variable BASIC A que nosotros hemos podido definir, ¡ni tan siquiera se le parece! Las diferencias son muchas; por ejemplo, esta «variable», que más propiamente se denomina REGISTRO, sólo puede contener un número natural entre 0 y 255.

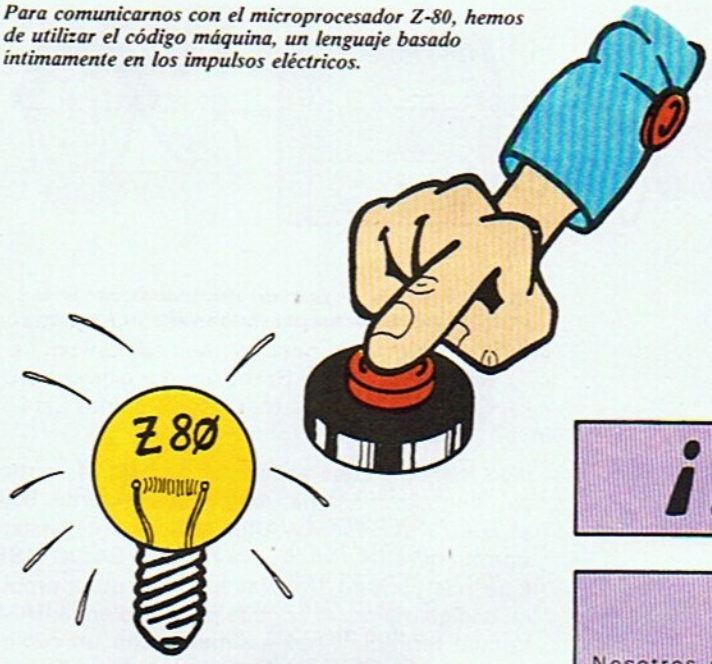
Está claro que las «variables» del código máquina, son bastante menos útiles que las del BASIC, sin embargo programando en el lenguaje del microprocesador no disponemos nada más que de este tipo de «variables», y lo que es más, sólo disponemos de ocho de estos registros, todos ellos numéricos, y de ninguno de manejo con cadenas.

Del mismo modo que utilizamos las instrucciones básicas del BASIC para conseguir programas más complejos, se pueden combinar las instrucciones básicas del código máquina para confeccionar programas más complejos, aunque lógicamente, al partir de una base más pobre, llegar a los mismos resultados necesita en este último caso mayores esfuerzos.

Los conocimientos básicos del ordenador se encuentran almacenados en el firmware.



Para comunicarnos con el microprocesador Z-80, hemos de utilizar el código máquina, un lenguaje basado intimamente en los impulsos eléctricos.



PASO DEL BASIC AL CODIGO MAQUINA

De lo dicho hasta ahora se desprende que siempre, estemos o no ejecutando un programa en có-

¡!

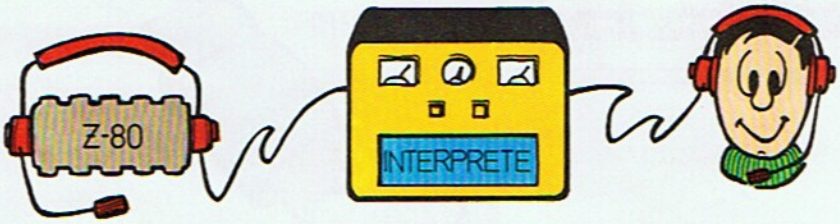
Nosotros podemos crear nuestras propias subrutinas en código máquina, depositándolas en cualquier zona de la memoria RAM.

*

Para la ejecución de una subrutina en código máquina, se utiliza la función BASIC USR, seguida de la primera dirección de memoria a interpretar.

*

Al ejecutar una subrutina en código máquina, puesto que no nos encontramos bajo el control del firmware, de nada servirá que intentemos detener el programa mediante **BREAK (CAPS SHIFT + SPACE)**.



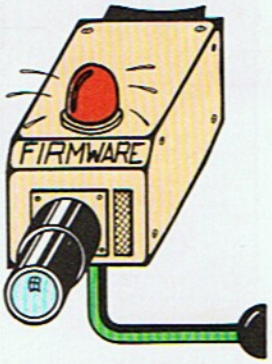
Gracias a la zona del firmware denominada intérprete, podemos comunicarnos con el ordenador en lenguaje BASIC.

digo máquina, nos encontramos bajo el control del lenguaje máquina; más concretamente, bajo el control del FIRMWARE. No obstante, existe una manera de evitarlo; la función BASIC **USR** Esta nos permite ejecutar una subrutina propia en código máquina, situada por tanto en la RAM, la cual terminará habitualmente con un código 201, que como sabemos es un retorno al punto desde el cual se llamó la subrutina; en nuestro caso concreto, este punto de llamada fue el BASIC, lo que significará recuperar el control completo por parte del *firmware*.

El utilizar la función **USR** tiene sus ventajas y sus inconvenientes: por una parte, al ser el código máquina un lenguaje mucho más simple, el tiempo de ejecución es considerablemente menor, es decir, un programa confeccionado en código máquina siempre funciona más rápido que uno realizado en BASIC y que cumpla un cometido idéntico.

Por otro lado, cada vez que algo no funciona bien, estando bajo el control del *firmware,* recibimos un mensaje de error, bastante documentado. Sin embargo, si no tenemos mucho cuidado, y contemplamos todas las posibilidades de error (cosa por otra parte no del todo fácil), puede ser que ante circunstancias extrañas el código máquina nunca regrese al BASIC, con lo cual nuestra única solución será apagar y encender el ordenador, para recuperar nuevamente el control.

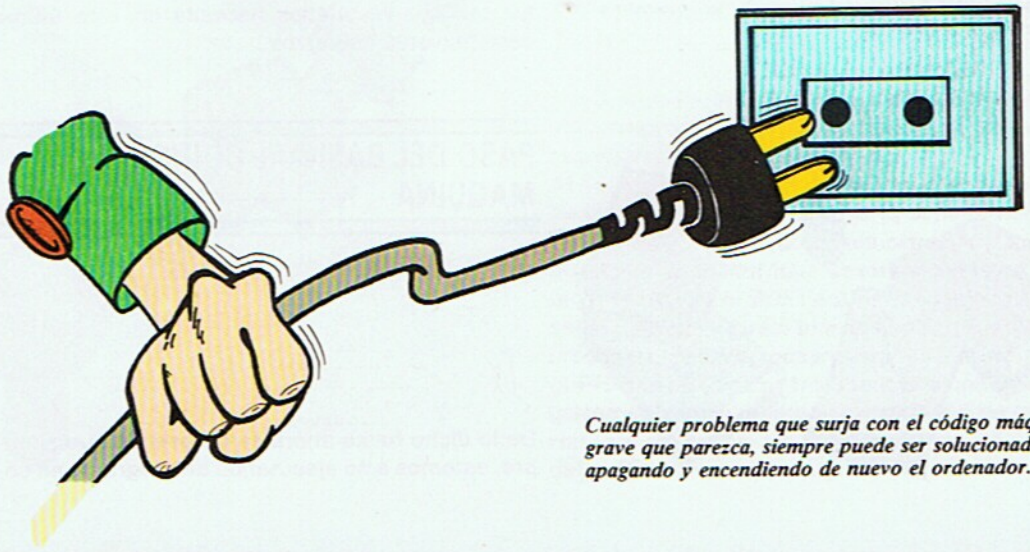
No obstante, en estos casos perdemos la información que hubiera en la memoria, y por tanto, es conveniente que en prevención de posibles problemas, antes de ejecutar un programa en código máquina, conservemos una copia en cinta del mismo. En todo caso, las posibles «catástrofes» que produzca el código máquina, al igual que las del BASIC, nunca llegarán más allá de tener que apagar y encender el ordenador, jamás le producirán un deterioro permanente; así pués ¡sin miedo! ¡experimentemos!



El firmware vigila constantemente nuestras operaciones en BASIC, procurando facilitarnos el trabajo.

CONTINUARA...

Pronto adquiriremos ya las nociones suficientes como para no tener ningún miedo al código máquina, e incluso haremos algún que otro miniprograma. Hasta entonces conviene que intentemos asimilar bien los conocimientos hasta ahora expuestos sobre el tema, puesto que serán la llave para abrir la puerta de este nuevo lenguaje. Por último, adelantaremos que se van a tocar conceptos de gran interés, como la REUBICABILIDAD, o el significado del ENSAMBLADOR, desentrañando ciertos misterios sobre el código máquina que aún permanecen sin desvelar.



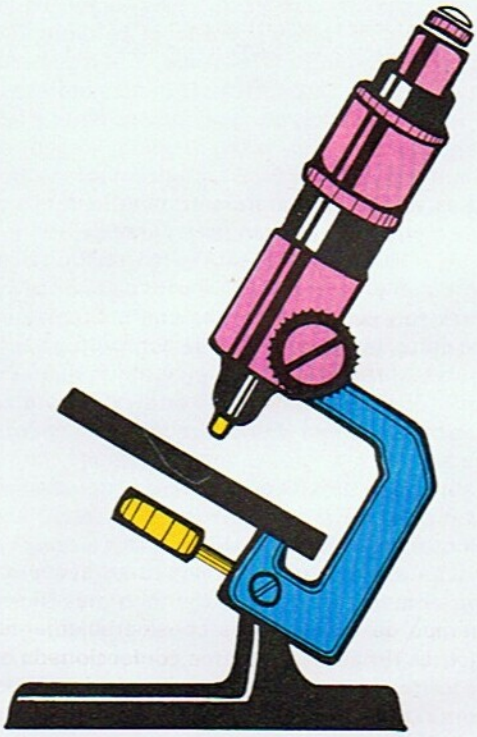
Cualquier problema que surja con el código máquina, por grave que parezca, siempre puede ser solucionado apagando y encendiendo de nuevo el ordenador.

CARACTERES GIGANTES

EN multitud de ocasiones, nuestros propios programas se ven necesitados de alguna subrutina capaz de potenciar la estética de los mismos, así como acentuar ostensiblemente la legibilidad de los mensajes o informaciones que aparecen en ellos.

A pesar de que ya hemos hecho uso de esta subrutina con anterioridad, es muy interesante disponer de un programa que nos explique la forma en que podemos incluirla en nuestros programas; sus espectaculares efectos habrán causado sensación entre los jugadores de LA BOMBA, y algunos otros programas. Todo ello no ha sido más que un adelanto de lo que se puede conseguir con esta subrutina de la firma británica PSION Computers, que se incluye a modo gratuito con la casete de demostración de los ZX Spectrum. Las posibilidades de la subrutina son muy grandes, puesto que no es sólo posible generar literales de las dimensiones que hemos visto en los mencionados programas, sino seleccionar a nuestro gusto su anchura, su altura y su punto de posicionamiento en la pantalla, teniendo este sistema de situación precisión de *pixel*.

Veamos a continuación la forma de experimentar con el programa de demostración. Ello nos servirá para pasar a continuacón a adaptar la subrutina a nuestros propios programas.

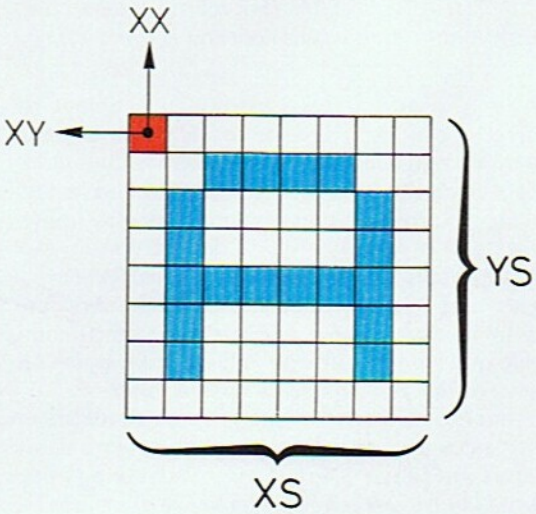


La subrutina de caracteres gigantes, permite aumentar a nuestro gusto las dimensiones de los caracteres estándar.

TRABAJANDO CON EL PROGRAMA

En primer lugar, deberemos introducir correctamente todas las líneas BASIC que componen el programa, y a continuación, seguir el sistema explicado en el programa LA BOMBA para la adopción de la subrutina en código máquina. Es recomendable antes de la primera ejecución de cualquier programa que incluya código máquina, conservar una copia grabada del mismo, puesto que en caso de haber cometido algún error de introducción, los resultados pueden ser catastróficos. Una vez ejecutado el programa, comenzará en la

Los factores que intervienen en la representación de caracteres son: XX, YY, XS e YS.



¡!

Tanto los números de instrucción que se utilizan para el soporte BASIC del código máquina, como las variables utilizadas para indicar la información referente al rótulo, pueden ser alteradas según las necesidades del programa concreto a que se aplique.

*

Indicando como factores de proporcionalidad vertical y horizontal (YS y XS) el valor uno, es posible emular los caracteres estándar del Spectrum, con la ventaja adicional de poder situarlos en cualquier punto de la pantalla.

¡!

Aunque nuestro modelo sea de 48 K, el programa de demostración está preparado para funcionar sólo con la subrutina para 16 K.

*

La dirección de carga, así como la de ejecución, ha de ser exactamente la especificada en el programa.

pantalla una secuencia de entrada de informaciones, mediante sentencias **INPUT.** Los datos a teclear son los siguientes:

1. ALTURA DEL CARACTER: el número introducido indicará las posiciones de carácter que tendrá el literal deseado en vertical. Dicho número ha de estar comprendido por tanto entre 1 y 21. Dado que la altura de un carácter estándar es la unidad, podemos decir que este factor, equivale al número de veces por el cual se multiplica la altura del mensaje (2 el doble, 3 el triple, etc...).
2. ANCHURA DEL CARACTER: este dato es muy similar al anterior, aunque concerniente a las dimensiones en horizontal; así pues, deberá estar comprendido entre 0 y 31. Lógicamente, este factor es equivalente al número por el cual se multiplica la anchura estándar del mensaje.
3. ALTURA DEL LITERAL: determina la posición (en *pixels),* en la cual se escribe el punto superior izquierdo del primer carácter a representar; por tanto, los valores han de estar comprendidos entre 0 y 192. Tengamos en cuenta, que al contrario que las coordenadas de **PLOT,** este parámetro toma como punto cero la línea superior de la pantalla.
4. LITERAL CENTRADO ?: como su nombre indica, centrará automáticamente el rótulo en horizontal. En el caso de que no aceptáramos esta opción, el ordenador nos pedirá el *pixel* que define el margen izquierdo del mensaje. Dicho pixel es el perteneciente a la esquina superior izquierda del primer carácter del rótulo a mostrar, y a partir del cual se imprime en pantalla todo el literal.
5. COLOR DE LA TINTA: imprimirá el rótulo con el color de tinta deseado.
6. COLOR DEL PAPEL: representará el mensaje con el fondo indicado.
7. BRILLO ?: imprimirá el mensaje con o sin brillo, dependiendo de la opción escogida.
8. FLASH ?: presentará el mensaje estable o con intermitencia, según nuestra indicación.
9. INTRODUCE LITERAL: por último, teclearemos el rótulo que queramos imprimir en la pantalla.

En lo referente a todos estos datos, hemos de destacar que cada uno de los literales suceptibles de ampliación, ha de ser cuidadosamente estudiado, con el fin de que su aparición en la pantalla sea correcta. En este sentido, es muy importante que el literal no salga del margen inferior de la pantalla, puesto que ello produciría la prácticamente segura «caída del Sistema». En horizontal no tenemos por qué tener esa precaución, dado que simplemente se sobreescribirán los caracteres del mensaje en la misma línea.
En cuanto a los **INPUT** de la toma de datos en los cuales se nos propone alguna opción, deberemos introducir SI o NO en respuesta a la pregunta que se nos realiza. Una vez finalizada la secuencia de entrada de datos, obtendremos un resumen de la misma, destinada a la comprobación de los valores tecleados. Si éstos no fuesen los correctos, se lo haríamos saber al programa pulsando **N,** con lo cual retornaríamos a la secuencia de introducción.

ADAPTACION A NUESTROS PROGRAMAS

Para el acceso de la subrutina de utilidad en CODIGO MAQUINA es necesario un pequeño soporte BASIC, cuyo listado aparece a continuación:

```
10 LET I=23306: POKE I,XX: POKE I+1,YY: POKE I+2,XS: POKE I+3,YS
20 POKE I+4,8: LET I=LEN P$: FOR K=1 TO I: POKE 23310+K, CODE P$(K): NEXT K
30 POKE 23311+I,255: LET I=USR 32256
```

La numeración de dichas instrucciones puede ser alterada por nosotros, con el fin de incluirla en la zona oportuna de nuestros propios programas. Dentro de la citada subrutina BASIC, las variables utilizadas tendrán los siguientes valores:

YS= FACTOR MULTIPLICADOR DE ALTURA.
XS= FACTOR MULTIPLICADOR DE ANCHURA.
YY= PIXEL DE COMIENZO EN ALTURA.
XX= PIXEL DE COMIENZO DE ESCRITURA EN HORIZONTAL.
P$= ROTULO A VISUALIZAR.

Los poseedores del modelo de 16 K, no tendremos más que incluir a continuación de todos los programas que utilicen la subrutina, el código máquina, tal como se explica en el programa la BOMBA.
Este sistema funcionará también en el modelo de 48 K, si bien es cierto que al situarse la subrutina a partir de la dirección 32256, limita la capacidad de nuestra memoria. Si queremos adaptar el código máquina a las 48 K de que disponemos, deberemos realizar las siguientes operaciones:

— Utilizar un programa cargador diferente al facilitado para el programa la Bomba.

```
10 REM CARGADOR DE CODIGO MAQUINA
20 CLEAR 64868
30 FOR I=64869 TO 65145
40 INPUT A
```

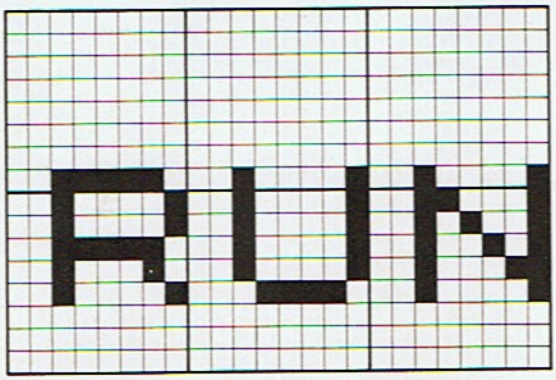
```
50 POKE I,A
60 NEXT I
```

— Efectuar estos POKEs.

POKE 64955,104	64956,253
POKE 64975,133	64976,253
POKE 64996,9	64997,254
POKE 65023,149	65024,253
POKE 65120,1	65121,254

Estos adaptarán nuestra subrutina casi al tope de memoria: 64860. Por tanto, la dirección de llamada (**USR**) que aparece al final de la subrutina BASIC, deberá ser también 64869. Del mismo modo, la protección mediante **CLEAR**, como se puede observar al comienzo del nuevo cargador, será 64868.

De entre los muchos efectos curiosos que se pueden obtener con la subrutina, destacaremos dos: por una parte, indicando XS=1 e YS=1, se simulan las dimensiones normales de los caracteres



Dando valor uno a los factores XS e YS, podemos escribir caracteres estándar «a caballo» entre varias posiciones de pantalla.

del Spectrum, con la ventaja de poder seleccionar por *pixels* el comienzo de escritura de los mismos, tanto en vertical como horizontal; esto nos permitirá escribir mensajes «a caballo» sobre varias posiciones de carácter, o incluso escribir en lugares difícilmente accesibles por el BASIC, como son las líneas reservadas al Sistema, y aún más, la línea de separación entre ambas zonas de la pantalla.

Por último el **POKE I+4,8** que señalamos al comienzo de la línea 20 en el soporte BASIC sugerido, indica la separación en puntos entre los caracteres consecutivos de un mensaje a tamaño normal. Variar este valor, puede suponer conseguir mensajes montados, o con mayor separación entre letras, así como escritura inversa (de derecha a izquierda), en caso de utilizar valores negativos.

¡!

Es muy importante elegir cuidadosamente las dimensiones y punto de comienzo del rótulo, para que éste no exceda los límites de pantalla por su zona inferior.

*

Recordemos cargar la subrutina en código máquina antes de ejecutar el programa de demostración, de no hacerlo así, sin duda tendremos que volver a cargar el programa.

```
  10 REM ************************
  20 REM * J.M.MAYORAL SERRANO *
  30 REM ************************
  40 REM * CARACTERES GIGANTES *
  50 REM ************************
  60 REM  PROGRAMA PRINCIPAL
  70 LET SW=0: DIM D(4): GO SUB 3440
  80 DATA "ANCHURA DEL CARACTER ...= ","ALTURA DEL CA
RACTER ....= ","ALTURA DEL LITERAL .....= ","LITERAL
CENTRADO .......= "
  90 DATA "COLOR DE TINTA ......= ","COLOR DEL PAPEL
.....=  ","BRILLO ..................= ","FLASH .......
...........= "
 100 RESTORE 80
 110 FOR N=1 TO 3
 120 READ A$
 130 PRINT
 140 PRINT TAB 1;A$;D(N)
 150 NEXT N
 160 READ A$
 170 PRINT
 180 PRINT TAB 1;A$; PAPER 2; INK 6; FLASH 1;B$
 190 PRINT
 200 IF B$="NO" THEN  PRINT TAB 1;"SEPAR. MARGEN IZQD
O ....= ";XX: GO TO 220
 210 PRINT
 220 GO SUB 3790
 230 PRINT : PRINT
 240 PRINT TAB 6; FLASH 1;" TODO CORRECTO ??"
 250 PRINT TAB 12;"(S/N)"
 260 IF INKEY$="N" THEN  CLS : GO TO 70
 270 IF INKEY$="S" THEN  GO TO 290
 280 GO TO 260
 290 CLS : PAPER P: INK T: FLASH FLASH: BRIGHT BRILLO
 300 GO SUB GS
 310 IF INKEY$="" THEN  GO TO 310
 320 BORDER 0
 330 PAPER 0
 340 INK 9
 350 BRIGHT 0
 360 FLASH 0
 370 RUN
3000 REM SBR. CARACT. GIGANTES
3010 LET xx=(256-8*xs*LEN p$)/2
3020 LET i=23306
3030 POKE i,xx: POKE i+1,yy: POKE i+2,xs: POKE i+3,ys
: POKE i+4,8
3040 LET i=i+4: LET u=LEN p$
3050 FOR q=1 TO u
3060 POKE i+q,CODE p$(q)
3070 NEXT q
3080 POKE i+u+1,255
3090 LET u=USR 32256
3100 RETURN
3110 REM  AUTO-EJEC. OBLIGATORIA
3120 BORDER 0: PAPER 0: CLS
3130 CLEAR 32255
3140 LOAD *"M";1;"LIT"CODE : POKE 23658,8
3150 REM  PRESENTACION
3160 LET P$="CARACTERES"
3170 LET XS=3
3180 LET YS=2
3190 LET YY=0
3200 BRIGHT 1: INK 6: PAPER 2
3210 GO SUB 3000
3220 LET YY=16
3230 LET YS=17
3240 LET XS=4
3250 PAPER 0
3260 LET P$="GIGANTES": INK 4
3270 GO SUB 3000
3280 FOR N=0 TO 200
3290 BORDER 1: BORDER 5: BORDER 6
3300 NEXT N
3310 LET P$="$ ZX Spectrum $"
3320 LET YY=160
3330 LET XS=2
3340 LET YS=1
3350 PAPER 1: INK 7: FLASH 1
3360 GO SUB 3000
3370 BORDER 0
3380 FOR N=0 TO 50
3390 BEEP .01,10: BEEP .02,20
3400 BEEP .03,30: BEEP .04,40
3410 NEXT N
3420 FLASH 0: PAPER 0: BRIGHT 0
3430 RUN
3440 REM  INTRODUCCION DATOS
3450 INPUT  PAPER 6; INK 9;"ALTURA CARACTER = ";YS: L
ET D(1)=YS
3460 INPUT  PAPER 6; INK 9;"ANCHURA CARACTER = ";XS:
LET D(2)=XS
3470 INPUT  PAPER 2; INK 9;"ALTURA LITERAL = ";YY: LE
T D(3)=YY
3480 INPUT  PAPER 2; FLASH 1; INK 9;"LITERAL CENTRADO
 (S/N)  "; LINE S$
3490 IF CODE S$=83 OR CODE S$=78 THEN  GO TO 3510
3500 BEEP 2,-10: GO TO 3480
3510 IF S$="N" THEN  INPUT "SEPARACION MARGEN IZQDO="
;XX: LET GS=3020: LET B$="NO": GO TO 3530
3520 LET B$="SI": LET GS=3000
3530 INPUT  PAPER 1; INK 9;"COLOR TINTA = ";T: GO SUB
 3670
3540 IF T>7 OR T<0 THEN  GO TO 3530
3550 INPUT  PAPER 1; INK 9;"COLOR PAPEL = ";P: LET SW
=1: GO SUB 3670
3560 IF P>7 OR P<0 THEN  GO TO 3550
3570 INPUT  INK 6;" BRILLO ??    (S/N)    ";C$
3580 IF C$="S" THEN  LET BRILLO=1: LET C$="SI": GO TO
 3610
3590 IF C$="N" THEN  LET BRILLO=0: LET C$="NO": GO TO
 3610
3600 GO TO 3570
3610 INPUT "FLASH ??     (S/N)   ";F$
3620 IF F$="S" THEN  LET F$="SI": LET FLASH=1: GO TO
3650
3630 IF F$="N" THEN  LET F$="NO": LET FLASH=0: GO TO
3650
3640 GO TO 3610
3650 INPUT  FLASH 1; PAPER 2; INK 6;"   INTRODUCE  LI
TERAL  ";P$
3660 RETURN
3670 REM
3680 REM
3690 DATA "NEGRO","AZUL","ROJO","MAGENTA","VERDE","CI
AN","AMARILLO","BLANCO"
3700 RESTORE 3690
3710 FOR N=0 TO 7
3720 READ M$
3730 IF SW=1 THEN  IF N=P THEN  LET E$=M$
3740 IF N=T THEN  LET I$=M$
3750 NEXT N
3760 RETURN
3770 REM
3780 REM
3790 RESTORE 90
3800 FOR N=1 TO 4
3810 READ A$
3820 PRINT
3830 PRINT TAB 1;A$
3840 NEXT N
3850 LET SW=0
3860 PRINT AT 11,24;I$
3870 PRINT AT 13,24;E$
3880 PRINT AT 15,27;C$
3890 PRINT AT 17,27;F$
3900 RETURN
```

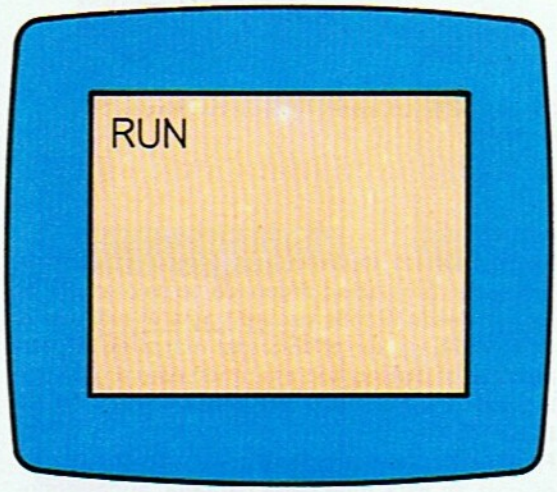
COLOR EN NUESTRO T.V.

EL color es uno de los atributos que hacen del Spectrum un ordenador atractivo y versatil, manteniendo una extraordinaria sencillez de manejo. Hasta ahora no sabemos hacer uso de esta importante herramienta, pero en el transcurso de este capítulo y el siguiente, perfilaremos los conocimientos necesarios como para empezar a explotar las posibilidades cromáticas de nuestro aparato, sacándole partido a esta nueva e importante faceta del mismo.

Lo primero que debemos saber sobre este tema es que, a efectos de color, la pantalla se encuentra dividida en dos zonas bien diferenciadas, cuya denominación podría ser: «primer» y «segundo» plano, o zona circundante y zona central de la pantalla.

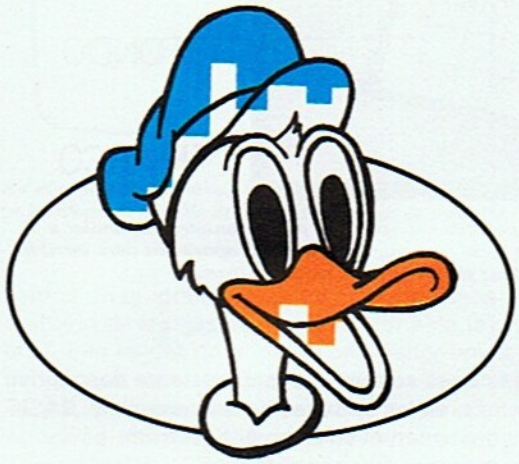
La zona de la pantalla sobre la cual podemos escribir, es una porción rectangular de 24 filas por 32 columnas. El resto de la misma, es decir, la parte que se encuentra por encima y por debajo, o a la izquierda y derecha de los márgenes citados, no es accesible a través de la sentencia **PRINT**.

La pantalla que se presenta en el televisor puede dividirse en dos zonas: exterior (no accesible por **PRINT***) e interior.*



ZONA ACCESIBLE POR PRINT

ZONA NO ACCESIBLE POR PRINT



A partir de este capítulo, podremos empezar a sacar provecho a otra de las cualidades de nuestro Spectrum: la generación de color.

Esta zona no utilizable para la escritura, conforma una especie de «fondo» o segundo plano, bordeando la pantalla, y puede adquirir cualquiera de los ocho colores básicos de que dispone el Spectrum, reseñados en la parte superior de las teclas correspondientes a los números.

Como podemos observar en la tabla adjunta, a cada uno de los colores le corresponde un número determinado, el cual utilizaremos para referirnos a él cuando programemos nuestro ordenador. Así por ejemplo, diremos que vamos a escribir en tinta 0, en vez de en tinta negra, o que vamos a emplear el color 2, en vez del color rojo, etc...

Lo mismo sucede con la parte central de la pantalla, con la ventaja de poder acceder a un segundo juego de colores, basado en el ya descrito. Este segundo juego de ocho colores se compone a partir de los básicos más un atributo denominado BRILLO. De esta forma, podemos decir que los ocho colores básicos son oscuros y los ocho restantes de tono más claro (más brillantes). Con esto hemos hablado de los dos colores de fondo posibles, el del marco de la pantalla, y el de la zona accesible por **PRINT**. Dentro de esta última, podemos distinguir aún dos sub-planos; conformando algo así como un primer plano y un «primerísimo» plano, siendo éste último el color en el cual escribimos realmente.

Llegados a este punto, quizá veamos las cosas

¡!

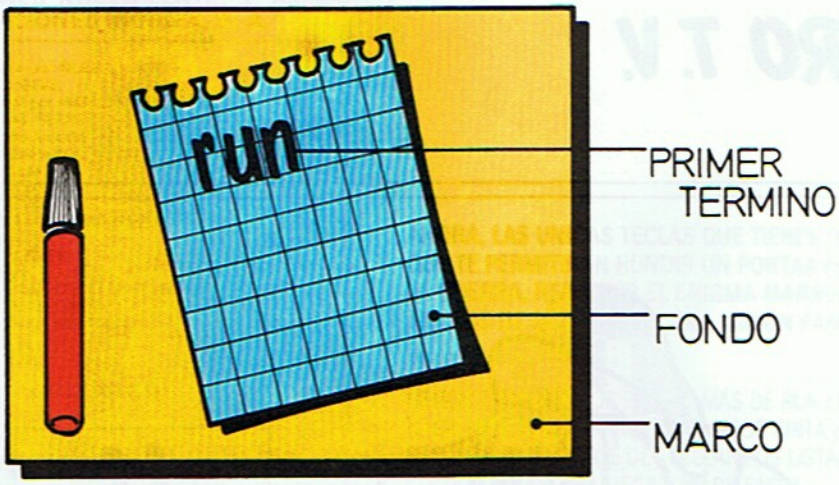
En la pantalla se pueden diferenciar dos zonas: la exterior, denominada marco, y en la cual no se pueden escribir caracteres, y la interior, donde se realizan las impresiones mediante **PRINT**.

*

Cada uno de los colores recibe un código numérico del 0 al 7.

*

BORDER altera el color del marco de la pantalla de forma inmediata.



La pantalla del Spectrum puede considerarse similar a una mesa (marco), en la cual nos apoyamos para escribir (primer término) sobre un papel (fondo).

más claras con un ejemplo, bastante descriptivo además, de los nombres de las sentencias BASIC que gestionan el color en el Spectrum.
Supongamos que nos disponemos a escribir una carta de forma manual. Para ello, nos proveemos de una mesa donde escribir, papel y bolígrafo. Por comparación de conceptos, podemos decir que el color de la mesa es el del marco de la pantalla, donde no podemos escribir; el del papel es el de la zona imprimible; y por último, el de la tinta del bolígrafo es el color del primerísimo plano anteriormente citado.

La sentencia **BORDER** *nos permite dar color al marco de la pantalla.*

LA SENTENCIA BORDER

Por medio de la sentencia **BORDER**, seguida de un código de color básico, es posible cambiar el tono del marco de la pantalla, el cual, al encendido del ordenador, adopta el blanco.
La gama de colores de este marco de pantalla, está restringida al ámbito de los ocho fundamentales. Veámos algunos ejemplos de los efectos que podemos conseguir, con su empleo, comenzando antes de cada ejercicio por teclear **BORDER 7 (marco blanco)** para restablecer el color inicial: **FOR I=0 TO 7: BORDER I: PAUSE 0; NEXT I**.

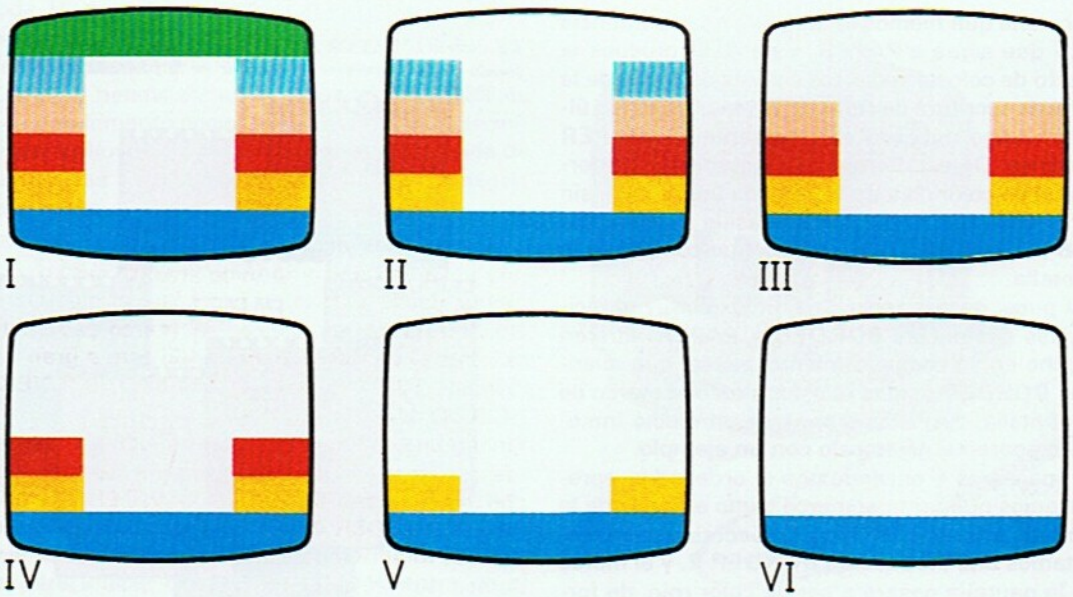
También es posible obtener marcos con franjas multicolor.

Esta línea nos muestra, por cada pulsación de una tecla, la relación ordenada de colores disponibles; desde el más oscuro hasta el más claro. Otra forma de ver el color del marco de la pantalla, es el conseguido alterando éste a gran velocidad: **10 FOR I=0 TO 7:BORDER I:NEXT I:GO TO 10.**
Finalmente, he aquí un último modo de obtener efectos en esta zona de la pantalla, aunque mucho más embarullados: **10 BORDER 2:BORDER 3:BORDER 4:BORDER 5:BORDER 6:GO TO 10.** Si este efecto de bandas de color no nos parece suficientemente sorprendente, podemos

probar a pulsar cualquier tecla, para verlo desplazarse de arriba abajo por la pantalla.
Estos efectos de color no se emplean muy a menudo, puesto que requieren una atención permanente del ordenador en esta tarea, lo cual resulta lógicamente bastante inútil. En todo caso, pueden ser empleados en combinación con otras sentencias BASIC que estudiaremos más adelante, para amenizar algunos tiempos de espera; por ejemplo, aquellos transcurridos desde la edición de una pantalla de instrucciones de un juego, hasta que la hemos leído y pulsamos una tecla para indicar esta circunstancia.
Sin duda, lo más frecuente es utilizar la sentencia **BORDER** del modo más simple estudiado al principio de este epígrafe. De esta forma, una vez que hemos asignado un determinado color al marco de la pantalla, éste permanece así hasta que indicamos lo contrario mediante una nueva sentencia **BORDER**. Explicaremos ahora brevemente el por qué del marco a franjas que hemos visto aparecer, por lo que puede ilustrarnos sobre el funcionamiento interno del ordenador en relación con el color del marco.
Cuando el televisor recibe la señal de vídeo del ordenador, comienza su impresión en la pantalla horizontalmente, de arriba a abajo, de una manera cíclica y constante. A este fenómeno se le conoce bajo el nombre de BARRIDO. El color del marco no es más que una de las señales que el ordenador envía al televisor, y así, si alteramos dicha señal de una forma lo suficientemente rápida, conseguiremos que al comienzo del barrido el marco sea de un color; un lapso de tiempo más tarde, antes de que finalice el barrido, de otro, y así sucesivamente hasta generar las franjas que hemos apreciado.
De hecho, si prestamos atención a lo que sucede en el marco de la pantalla cuando leemos o grabamos un programa en cinta, observaremos que multitud de franjas ocupan esta zona de la pantalla. Ello es debido a que el ordenador envía, al tiempo que lee y escribe, diversas señales al televisor, y puesto que es capaz de hacerlo a una velocidad mayor que la que nosotros conseguimos con **BORDER** desde el BASIC, puede generar un mayor número de franjas.
Aunque en principio el marco de la pantalla sea una zona poco interesante, dado que en ella no podemos escribir, no debemos desechar la posibilidad de su manejo. En el quinto capítulo de la sección de TU SPECTRUM, encontraremos más información sobre este tema.

En el proceso de formación de la imagen en el televisor, se produce un barrido horizontal cíclico de la pantalla.

Si en un corto espacio de tiempo (inferior a un barrido), enviamos varias señales diferentes como color de marco para la pantalla, este adopta en cada momento el correspondiente, generándose un marco a franjas.



¡!

Para que **PAPER** o **INK** afecte simultáneamente a toda la pantalla, hemos de ejecutar a continuación un comando **CLS**.

*

El brillo suplementario que se obtiene mediante la sentencia **BRIGHT** (en inglés, brillo), no afecta al marco de la pantalla, sino sólo a su zona interior.

*

Los atributos **BRIGHT** y **FLASH** (brillo e intermitencia), afectan simultáneamente a la **INK** (tinta) y el **PAPER** (papel) de un caracter.

*

Sólo **BORDER** tiene un efecto inmediato sobre la pantalla; el resto de los atributos de color afectan a los caracteres escritos a continuación, pero no a los ya presentes en la pantalla.

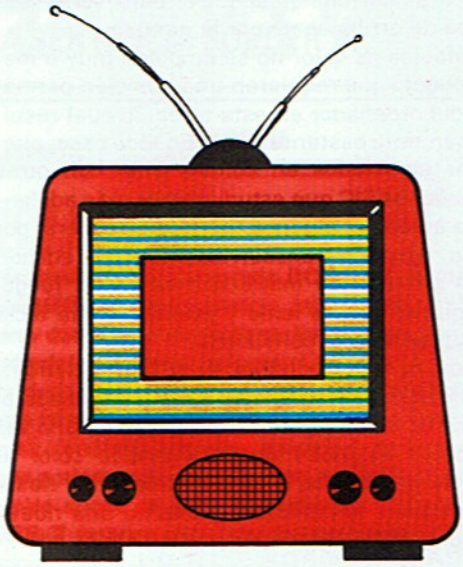
Al encender el ordenador, el estado original de los atributos de color es el siguiente: **BORDER 7, PAPER 7, INK 0, BRIGHT 0** y **FLASH 0.**

Si alteramos la señal de color enviada al marco en un tiempo inferior al de un barrido, podemos obtener marcos con franjas multicolor.

LA SENTENCIA PAPER

Del mismo modo que **BORDER** asigna color al marco de la pantalla, **PAPER** lo hace para el centro de la misma. Sin embargo, la sentencia **PAPER** es mucho más potente: permite asignar un color determinado a cada posición concreta de la pantalla, es decir, a cada caracter en particular, en dos modalidades de brillo, gracias a la combinación con la sentencia BASIC **BRIGHT**, estudiada más adelante.
La estructura de la sentencia **PAPER** es similar en todo a **BORDER**, y va seguida por tanto de un código de color. En el siguiente ejemplo, veremos las posibles combinaciones de colores de marco y fondo, empleando los fundamentales: **FOR I=0 TO 7: FOR J=0 TO 7: BORDER I: PAPER J: CLS: NEXT J: NEXT I.**

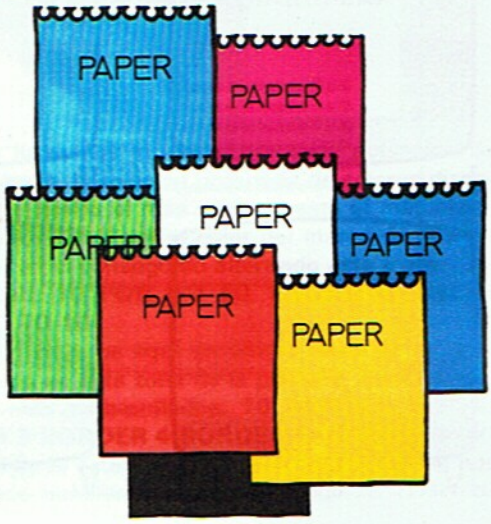
Las combinaciones de PAPER *y* BRIGHT *se pueden utilizar en cada caracter, independientemente del resto de la pantalla.*

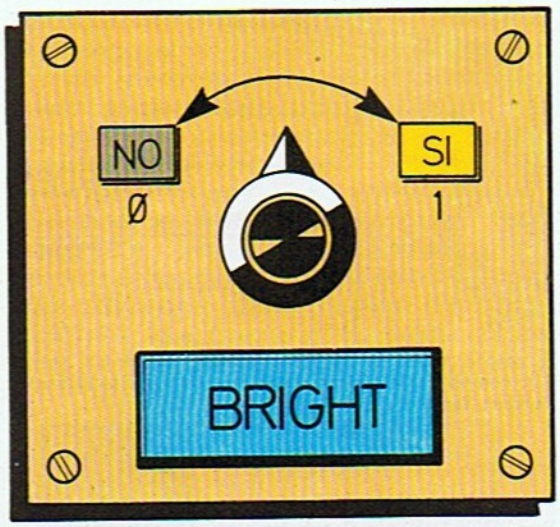


La extraordinaria velocidad del código máquina permite generar más franjas que las obtenidas desde el BASIC mediante BORDER. *Esto es lo que sucede cuando nuestro Spectrum lee o graba programas.*

Conviene que fijemos la atención en la sentencia **CLS** que sigue a **PAPER**. Este **CLS**, produce el efecto de colocar todos los colores de fondo de la zona de escritura de caracteres, en el valor del último código indicado en una sentencia **PAPER** genérica. De esta forma, conseguimos la uniformidad de color de toda la pantalla imprimible, sin tener que recurrir a colorear, casilla a casilla, las 768 (24 filas de 32 columnas), que componen la pantalla.
Así pues, pese a tener una sintaxis muy parecida, las sentencias **BORDER** y **PAPER** difieren mucho en su comportamiento, puesto que mientras **BORDER** asigna el color a todo el marco de la pantalla, **PAPER** no produce un efecto inmediato aparente. Veámoslo con un ejemplo.
Si apagamos y encendemos el ordenador, apreciaremos que tanto el marco como el fondo de la pantalla interior son blancos. A continuación ajecutamos una instrucción **BORDER 2**, y el marco de la pantalla pasará a ser de color rojo, de forma inmediata. Ahora ejecutemos **PAPER 2**. El mensaje **0 OK** se editará en la última línea de la pantalla, pero sin embargo el interior de la pantalla sigue siendo blanco ¿qué es lo que ha ocurrido? ¿acaso la instrucción no ha sido formulada correctamente?
En absoluto. El problema estriba en que el efecto de **PAPER** no es inmediato, como en el caso de **BORDER**, sino que afecta a las instrucciones de

La sentencia PAPER *nos permite asignar el color de fondo de los caracteres a escribir en la pantalla.*





Mediante **BRIGHT** *podemos conectar o desconectar el modo de brillo suplementario.*

manejo de pantalla sucesivas, como **PRINT, TAB, CLS,** etc... Probemos a escribir una letra A en la pantalla: **PRINT "A"**. Por fin hace su efecto la sentencia **PAPER**: la letra ha sido escrita sobre un fondo rojo.

En resumen, podemos afirmar que la sentencia **BORDER** indica al Spectrum que debe alterar el color de todo el marco de la pantalla de forma inmediata, mientras que **PAPER** le señala el color que debe emplear como fondo en las próximas operaciones de escritura.

Este hecho nos ha ayudado a descubrir también que el efecto de **CLS** no es sólo el de borrar la pantalla, como habíamos visto hasta ahora, sino que también establece como color de fondo para toda la pantalla el seleccionado con la última sentencia **PAPER** (blanco inicialmente, al encender el ordenador).

Como ya hemos dicho, la sentencia **BRIGHT** es un complemento opcional de **PAPER**, que permite dar brillo al color. Veamos una combinación de sentencias **PAPER** y **BRIGHT**, para conseguir una «carta de ajuste» de color de fondo.

```
 10 REM - AJUSTE DE COLOR
 20 FOR I=0 TO 1
 30 BRIGHT I
 40 FOR J=1 TO 10
 50 FOR K=0 TO 7
 60 PAPER K: PRINT '    ';
 70 NEXT K
 80 NEXT J
 90 NEXT I
100 BRIGHT 0
```

Como podemos apreciar en el listado, la sentencia **BRIGHT** lleva como parámetro un número que adopta los valores 0 ó 1. El primero se utiliza para indicar que el color se encuentra en el modo normal, y el segundo que posee brillo suplementario. Como en el caso de PAPER, las instrucciones **BRIGHT 0** y **BRIGHT 1**, tendrán efecto sobre las escrituras sucesivas, pero no sobre toda la pantalla.

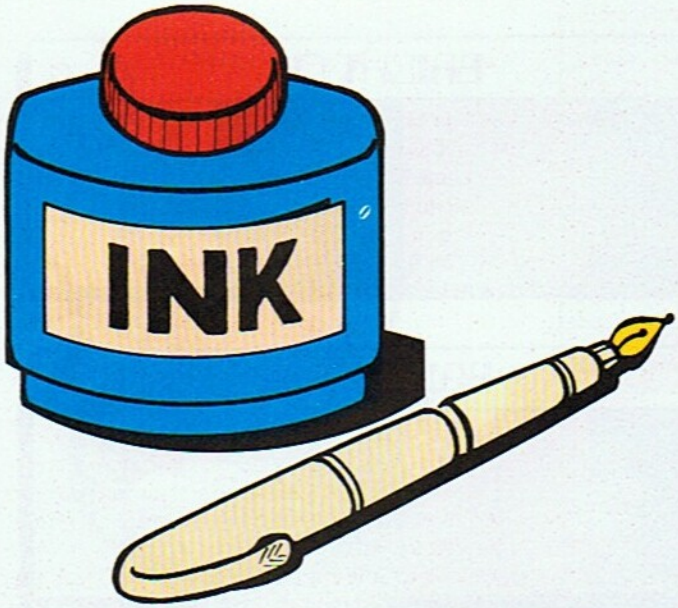
Finalmente, resta advertir que si utilizamos para la instrucción **BRIGHT** un parámetro menor que cero, obtendremos el mensaje de error **B Integer out of range** (entero fuera de rango); por el contrario, empleando parámetros por encima de 1, el mensaje emitido será **K Invalid colour** (código de color no válido). Una excepción a esto último será el código ocho, cuyo peculiar efecto estudiaremos posteriormente.

LA SENTENCIA INK

Hasta ahora hemos visto como dar color a la mesa sobre la que nos apoyamos (**BORDER**), y al papel en el cual escribimos (**PAPER**), por otro, nos falta conocer la manera de depositar la tinta adecuada en nuestro bolígrafo.

A través de la sentencia **INK** se define el color

La sentencia **INK** *nos permite alterar el color de primer término de los caracteres a escribir.*



¡!

Los parámetros para **BRIGHT** (brillo) y **FLASH** deben estar comprendidos entre 0 y 1, significando desconexión y conexión del modo, respectivamente.

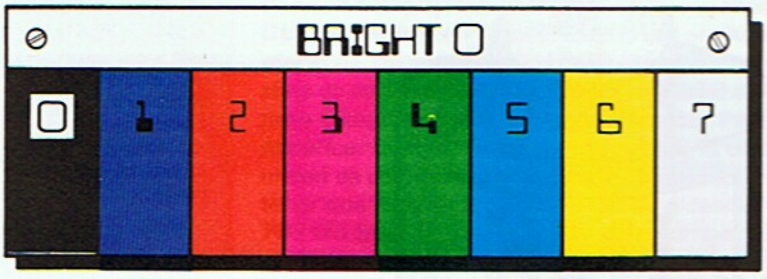
*

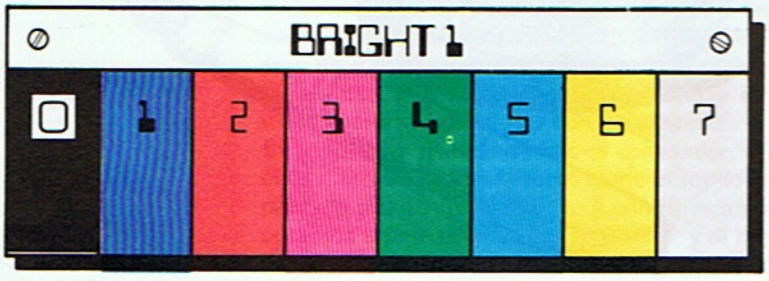
Las diferencias entre los tonos con y sin brillo suplementario de un mismo color, decrecen según se oscurece éste.

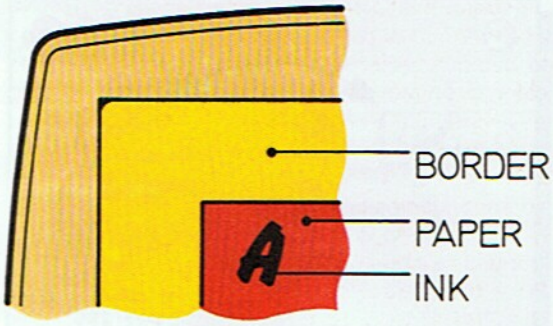
de esta tinta. De forma similar a casos anteriores, la sentencia va seguida de un parámetro que señala el código de color deseado. Como era de esperar, tanto su sintaxis como su funcionamiento es muy similar a **PAPER**.
Así pues, el cambio del color de tinta no afecta de forma inmediata a toda la pantalla, sino sólo a las impresiones que se efectúan a partir de la ejecución de la instrucción correspondiente. Ello nos permite escribir en cada posición de caracter (coordenada fila-columna), con el color de papel (en inglés *paper*) y tinta (en inglés *ink*) que deseemos, independientemente de los que se utilicen para las otras posiciones de la pantalla. Aprovechando parte del programa anterior, podemos escribir ahora uno que nos muestre una «carta de ajuste» completa, de combinaciones de colores de papel y tinta, con y sin brillo.

```
 10 REM - AJUSTE DE COLOR
 20 FOR I=0 TO 1
 30 BRIGHT I
 40 FOR J=1 TO 10
 50 FOR K=0 TO 7
 60 PAPER K: PRINT '    ';
 70 NEXT K
 80 NEXT J
 90 NEXT I
100 FOR I=0 TO 1
110 BRIGHT I: PAPER 7
120 FOR J=0 TO 3
130 INK J: PRINT J;'   ';
140 NEXT J
150 PAPER 0
160 FOR J=4 TO 7
170 INK J: PRINT J;'   ';
180 NEXT J
190 NEXT I
200 PAPER 7: BRIGHT 0: INK 0
```

Tanto para **PAPER** *como para* **INK**, *el rango de su parámetro oscila entre cero y siete, con dos modos de brillo.*







Los colores de **BORDER, PAPER** *e* **INK** *son totalmente independientes entre sí.*

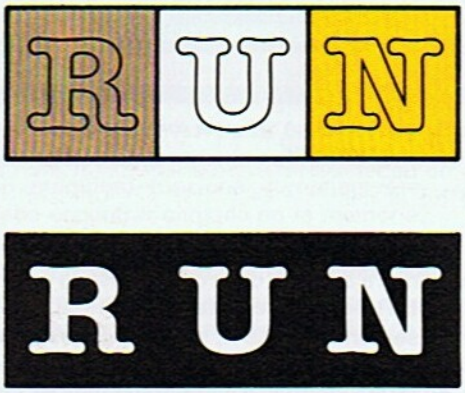
Por último, y en relación con el margen en el cual se mueven los parámetros para **INK** y **PAPER**; éstos no deben ser inferiores a cero, puesto que como en el caso de **BRIGHT**, se producirá un error del tipo **B Integer out of range** (entero fuera de rango).
En esta ocasión, los colores admitidos llegan hasta el siete, aunque a ellos hemos de añadir dos códigos más: el ocho y el nueve, cuyo efecto no es exactamente el de un color particular, sino algo más especial que estudiaremos en el próximo capítulo. Cualquier parámetro por encima de nueve, motivará la emisión del mensaje de error **K Invalid colour**, cuyo significado hemos tenido la oportunidad de conocer anteriormente.

CONSIDERACIONES GLOBALES

Si somos un poquito observadores, veremos que el orden de los códigos de color se corresponde con una secuencia desde los más oscuros hacia los más claros. De hecho, si anulamos por un momento el color de nuestra T.V., veremos que aparecen diferentes tonalidades de grises, ordenadas de menor a mayor luminosidad.
Del mismo modo, se hace patente que la diferencia entre los colores con y sin **BRIGHT** es muy acentuada en los tonos claros, y casi imperceptible en los oscuros. Este efecto debe ser tenido muy en cuenta, puesto que puede ser aprovechado a nuestro favor en alguna ocasión; dentro de este mismo apígrafe, cuando tratemos otra de las características de **BRIGHT**, nos daremos cuenta del por qué de esta afirmación.
Puesto que el marco de la pantalla sólo puede re-

presentar los ocho colores básicos, y no se ve afectado por el factor de brillo, podemos utilizarlo como referencia sobre la luminosidad de los colores empleados en la pantalla interior. Ello nos conducirá a curiosos efectos ópticos, como el que a continuación describimos.

Ya hemos comprobado anteriormente que cuando encendemos el ordenador, el color inicial para marco y fondo de pantalla es el blanco, encontrándose ambos en la modalidad de brillo normal. Pues bien, ejecutemos la siguiente instrucción: **BRIGHT 1: CLS.** Gracias a ella, conectaremos el modo de brillo suplementario, y borraremos la



Para resaltar el brillo suplementario de tinta, sin que aparentemente afecte al papel, debemos emplear fondos oscuros.

pantalla, de forma que el nuevo brillo afecte completamente al interior de la misma.

Casi como por arte de magia apreciamos que el cambio no se ha efectuado en el interior de la pantalla, como nosotros esperábamos, dado que sabíamos que **BRIGHT** no afecta al marco. Aparentemente, el fondo ha permanecido en su color blanco inicial, y el marco de la pantalla ha adquirido un tono gris, que en teoría se encuentra fuera de la gama del Spectrum, y más aún, de las posibilidades de esa zona de la pantalla.

En realidad, todo ha sido un simple efecto óptico. Como nosotros ya suponíamos, **BRIGHT** no ha afectado al marco, sino al fondo; pero el contraste del brillo suplementario entre uno y otro blanco, nos ha hecho creer que el primero de los colores, no es tal blanco, sino un perfecto gris. Este efecto cromático no debe ser desaprovechado, y combinado con inteligencia, puede «arrancarle» al Spectrum algún color más que aquellos de los cuales originalmente dispone, o al menos dar tal impresión.

Como hemos dicho unos párrafos más atrás, esta diferencia entre los tonos brillantes y no brillantes, decrece según se va oscureciendo la gama, y así como en el blanco está acentuadísima, hecho este que hemos podido comprobar tan solo hace unos instantes, en el negro es prácticamente imperceptible.

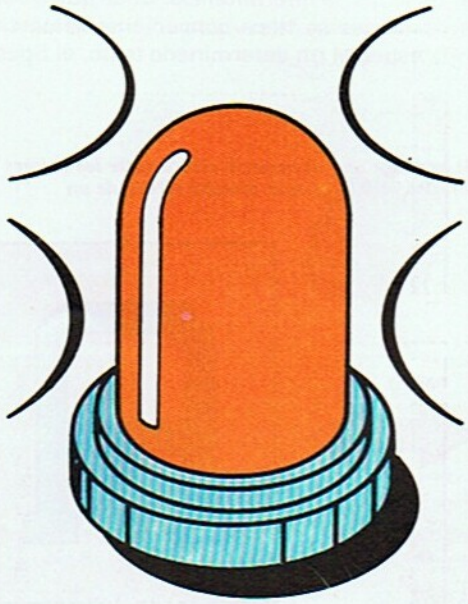
CODIGOS DE COLOR

0	NEGRO
1	AZUL OSCURO
2	ROJO
3	ROSA
4	VERDE
5	AZUL CLARO
6	AMARILLO
7	BLANCO

El siguiente mini-programa nos ayudará a comprobar esta escala decreciente de diferencias, gracias al contraste entre el color del marco (sin brillo), y el del interior (con brillo suplementario); pulsemos una tecla por cada vez que queramos pasar al próximo color: **BRIGHT 1: FOR I=7 TO 0 STEP -1: BORDER I: PAPER I: CLS: PAUSE 0: NEXT I.**

Otra característica a considerar, en lo referente al atributo de color **BRIGHT**, es que afecta tanto al papel como a la tinta que concurren en una determinada posición de la pantalla. Como ya vimos anteriormente, en una misma posición de la pantalla (coordenada fila-columna), podíamos hacer coexistir un color diferente de la tinta y del papel; sin embargo, si decidimos que dicha posi-

La sentencia **FLASH** *nos permite crear caracteres con colores intermitentes.*



¡!

Los colores de fondo y primer término que se otorguen a un caracter, así como los otros atributos (**BRIGHT** y **FLASH**), afectan a cada carácter de la pantalla, independientemente del resto de ellos y del marco de la misma.

*

El mensaje **B Integer out of range** (entero fuera de rango) aparece cuando queremos utilizar valores negativos para los atributos de color.

*

El mensaje **K Invalid colour** (código de color no válido) aparece cuando utilizamos como parámetro para los atributos de color, códigos que superan el nueve.

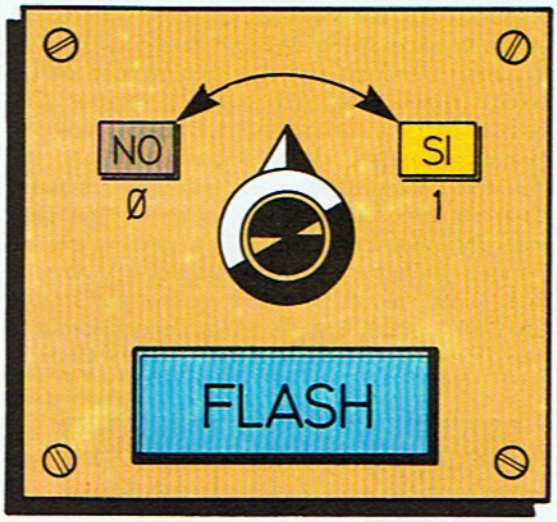
ción se encuentre con brillo suplementario, este hecho afectará tanto al color de la tinta como del papel en esa posición. Una vez que hemos visto esto, estamos en condiciones de explicar por qué dijimos que le podríamos sacar provecho a la diferencia decreciente entre los tonos brillantes y no brillantes de los diversos colores.

Supongamos que queremos escribir la letra inicial de una palabra en tono brillante, para destacarla del resto de la misma, pero sin embargo, nos interesa que permanezca de igual tono el fondo, por ejemplo, porque lo hemos hecho coincidir con el color del marco, para así aumentar las dimensiones aparentes de la pantalla (capítulo cinco, sección de TU SPECTRUM). Esta opción es aparentemente imposible, dado que el brillo afectará tanto a la tinta como al fondo (papel) del caracter.

La solución se encuentra en no utilizar en el programa un color de fondo claro, sino todo lo contrario: el color negro, o en su defecto el azul oscuro. Gracias a ello, aunque la cualidad de brillo suplementario atañe también al fondo, su efecto es prácticamente imperceptible, y no así en la tinta, para la cual habremos empleado un tono muy claro, a ser posible el blanco.

UN ULTIMO TRUCO

Pero además de las posibilidades de color descritas, contamos con una más, quizá la más espectacular de todas: la intermitencia. Dado que en algunas ocasiones se hace conveniente remarcar de forma especial un determinado texto, el Spectrum incorpora además de la característica de brillo (**BRIGHT**), la de intermitencia, obtenida mediante la sentencia **FLASH**, seguida de los parámetros 1 ó 0, para conexión o desconexión del modo, de forma idéntica a como funcionamos con **BRIGHT**.



Los parámetros para FLASH *están comprendidos entre 0 y 1: estable o intermitente.*

Los errores que se producen al no respetar estos parámetros son los mismos que los emitidos con **BRIGHT**, incluyendo la posibilidad de utilizar el código especial 8, que será objeto de estudio en el próximo capítulo.

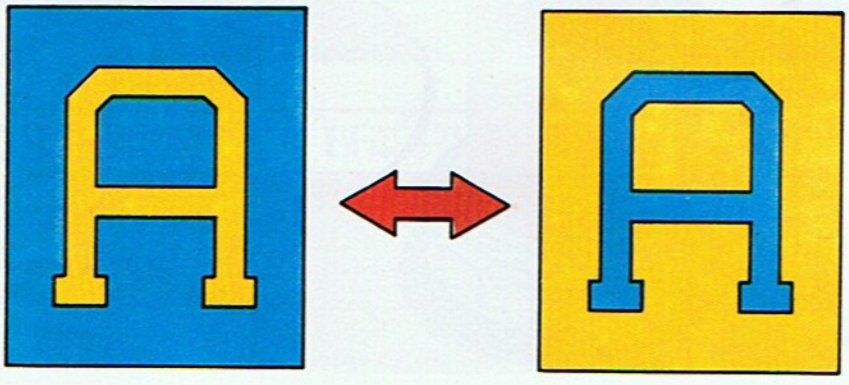
El efecto de **FLASH** es producir una alternancia intermitente entre los colores de tinta (**INK**) y fondo (**PAPER**) de un carácter, de forma automática, y con una velocidad constante. Con este nuevo aditamento, son muchas las combinaciones de color posibles dentro del área accesible de la pantalla; podríamos decir que casi infinitas. De hecho, tenemos que considerar lo siguiente:

PAPER.:	**Valores de 0 a 7**
INK.:	**Valores de 0 a 7**
BRIGHT:	**Valores de 0 a 1**
FLASH.:	**Valores de 0 a 1**

El número de combinaciones posibles es de 256, lo cual podremos comprobarlo a través de un último programa.

```
  10 REM - COMBINACIONES DE COLOR
  20 FOR I=0 TO 7: FOR J=0 TO 7: FOR K=0
 TO 1: FOR L=0 TO 1
  30 GO SUB 70: PAUSE 10
  40 NEXT L: NEXT K: NEXT J: NEXT I
  50 PAUSE 0: PAPER 7: INK 0: BRIGHT 0:
FLASH 0: CLS
  60 STOP
  70 PAPER I: INK J: BRIGHT K: FLASH L
  80 PRINT "PAP.: ";I;" INK.: ";J;" BRI.
: ";K;" FLA.: ";L;" "
  90 RETURN
```

FLASH *produce una alternancia cíclica entre los colores de fondo (*PAPER*) y primer término (*INK*) de un carácter.*



EL CODIGO MAQUINA

VAMOS ahora a hacer algunas precisiones a cerca de la carga y ejecución de programas en código máquina con el Spectrum; quizás algunos de nosotros ya las sepamos, pero en todo caso no está de más refrescar esos conocimientos.

Una subrutina en código máquina es un conjunto de números (bytes) situados en determinada zona de la memoria, a los cuales se accede mediante la función **USR**. Así pues, para que un programa que utilice determinada subrutina en C/M (código máquina) funcione correctamente, esta se debe encontrar cargada en la memoria.

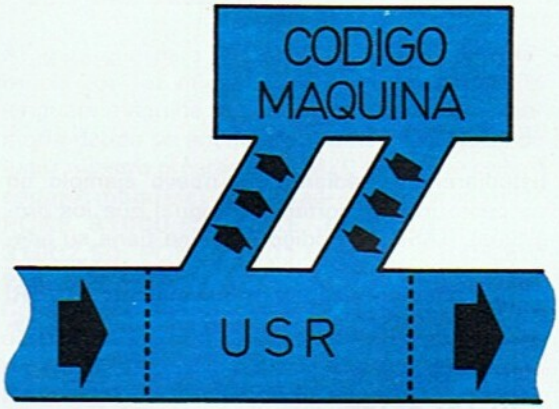
Al desconectar nuestro aparato, o al pulsar el botón de *reset* en el PLUS, la memoria RAM se borra íntegramente. Esto quiere decir, que no es suficiente introducir el código máquina una sola vez, sino siempre que vayamos a cargar el programa que la necesite.

Veamos un ejemplo. Vamos a hacer un pequeño programa en código máquina; realmente, lo único que hace es retornar al BASIC. Así pues, la subrutina va a ser extraordinariamente corta, dado que sólo estará compuesta por el código 201. Este será depositado en la dirección 30000.

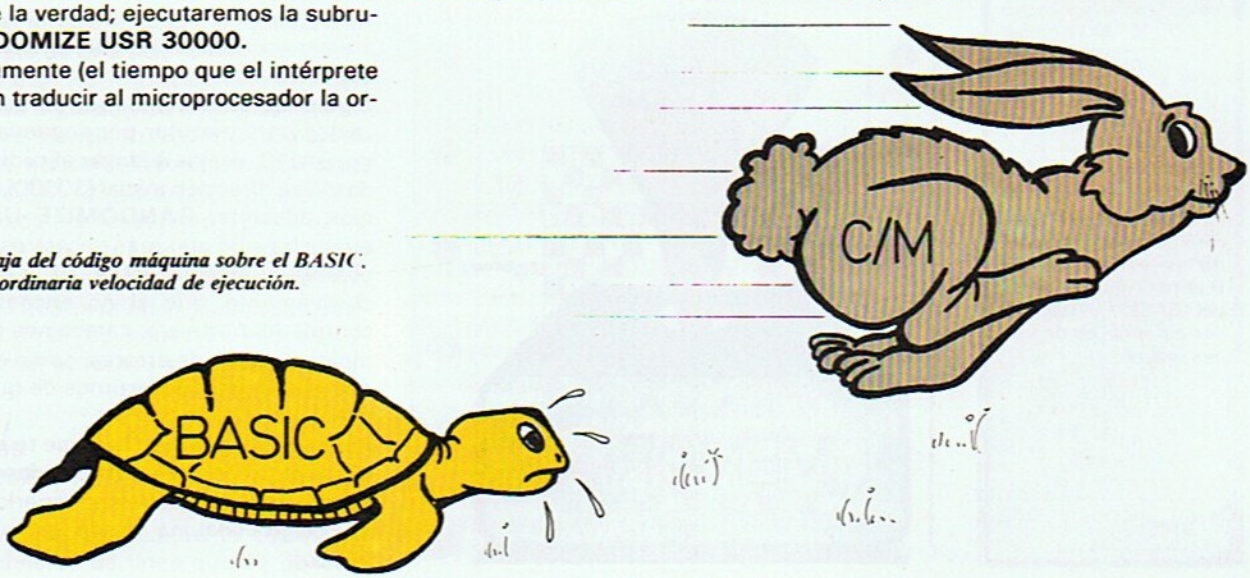
Para introducir en la memoria tan corto programa, sólo es necesario ejecutar la siguiente instrucción: **POKE 30000,201**. Gracias a ella, situamos nuestra mini-subrutina en la dirección 30000. Para comprobar que sigue allí, no tenemos más que preguntar el contenido de esa dirección (**PRINT PEEK 30000**). Bien, ahora llega el momento de la verdad; ejecutaremos la subrutina con **RANDOMIZE USR 30000.**

Casi inmediatamente (el tiempo que el intérprete BASIC tarda en traducir al microprocesador la orden), recibiremos el mensaje **0 OK**. Acabamos de terminar con éxito nuestro primer programa en código máquina, si bien es cierto que no tiene mucha utilidad (simplemente vuelve al BASIC). Ahora bien, él nos va a dar oportunidad de comprobar en la práctica que su efecto desaparece al apagar el ordenador.

Ahora que ya hemos vuelto a encender, ejecute-



La función USR produce una bifurcación temporal del control BASIC hacia una subrutina en código máquina.



La principal ventaja del código máquina sobre el BASIC, reside en su extraordinaria velocidad de ejecución.

¡!

Un programa en código máquina sólo puede ser ejecutado desde la dirección de memoria destinada a tal fin.

*

Un programa se denomina reubicable cuando cualquiera que sea la dirección de memoria en que se sitúe, funciona correctamente, siempre y cuando sea ejecutado desde la dirección adecuada.

*

La reubicabilidad no es un hecho común, y prácticamente sólo se da en rutinas relativamente cortas en longitud y complejidad.

*

La velocidad de ejecución del código máquina es extraordinariamente superior a la del BASIC, y llega a ser centenares de veces mayor.

mos de nuevo la instrucción **RANDOMIZE USR 30000**. Esta vez no ha ocurrido lo mismo, y ello se debe a que en la dirección 30000 ya no se encontraba nuestra subrutina en código máquina

CONCEPTOS IMPORTANTES

Estudiaremos mediante un nuevo ejemplo un concepto muy importante. Al igual que los programas BASIC, el código máquina tiene su adecuado punto de inicio, y si queremos que funcione correctamente, tenemos que ejecutarlo (**USR**) a partir de éste. Observemos el siguiente programa BASIC:

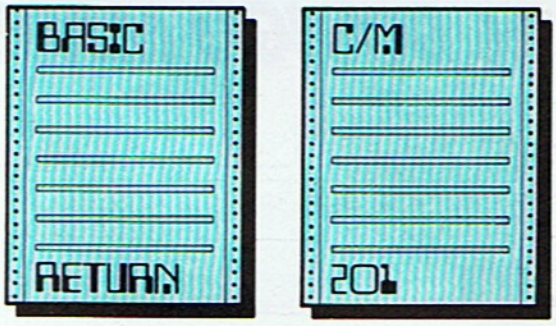
```
10 LET A=199
20 POKE 22528,A
```

Su misión es encender en la esquina superior izquierda de la pantalla, un cuadrado de color negro y blanco en intermitencia. Si lo ejecutamos mediante **RUN** o **RUN 10**, que en este caso es lo mismo puesto que 10 es la primera línea de instrucción, cumplirá perfectamente su cometido; sin embargo, ejecutado mediante **RUN 20**, producirá un error del tipo **2 Variable not found.** La traducción de este programa al lenguaje máquina será algo más complicada que un mero código 201, pero no mucho más. Supongamos que como en el caso anterior, deseamos ubicar la subrutina a partir de la dirección 30000; de ser así, la introducción del programa se conseguirá mediante la siguiente línea de instrucción: **POKE 30000,62: POKE 30001,199: POKE 30002,50: POKE 30003,0: POKE 30004,88: POKE 30005,201.**

Ejecutando este mini-programa con **RANDOMIZE USR 30000**, podremos ver que se obtiene un efecto idéntico al observado con la ejecución del último listado BASIC; es decir, la aparición de un cuadrado en intermitencia (negro y blanco), en la posición superior izquierda de la pantalla. Pero las similitudes entre estos dos programas, pese a estar escritos en lenguajes diferentes, no acaban ahí.

Siguiendo un sistema muy parecido al que utilizamos para ejecutar por segunda vez el programa BASIC, vamos a lanzar el código máquina, no desde su dirección inicial (30000), sino desde una más adelante: **RANDOMIZE USR 30001**. El efecto ha sido aun más desastroso que el que se obtenía al ejecutar **RUN 20**, y ello se debe exclusivamente, que al no encontrarnos bajo el control del *firmware*, carecemos de un mecanismo de emisión de errores, como el de **2 Variable not found**, para advertirnos de que algo no marcha del todo bien.

Así pues, al igual que hay que tener cuidado para ejecutar un programa BASIC desde el lugar adecuado, aún más debemos tenerlo, para ejecutar el código máquina desde su punto exacto de inicio.



Existen diversas similitudes entre los lenguajes BASIC y código máquina.

La sentencia **POKE** *se utiliza para la introducción por el teclado de programas en código máquina.*

Los programas, tanto en BASIC como en código máquina, deben ejecutarse desde la línea para la que están preparados.

REUBICABILIDAD

Ahora vamos a utilizar un nuevo concepto: la REUBICABILIDAD; aplicable tanto al BASIC como al C/M (código máquina), pero que como en el último ejemplo estudiado, tiene aún más importancia al tratarse en lenguaje máquina, puesto que es más susceptible de producir «desastres».
Un programa se dice REUBICABLE cuando cualquiera que sea su situación en la memoria, o su numeración de referencia, funciona igualmente. Con un ejemplo quedará más clara esta idea. En el caso del BASIC, la numeración de referencia está constituida por los números de línea de instrucción; por tanto, un programa BASIC que funcione correctamente cualquiera que sean los números de línea que se utilicen, siempre y cuando se respete el orden de ejecución, es reubicable. El programa del cuadrado intermitente ejemplo es reubicable:

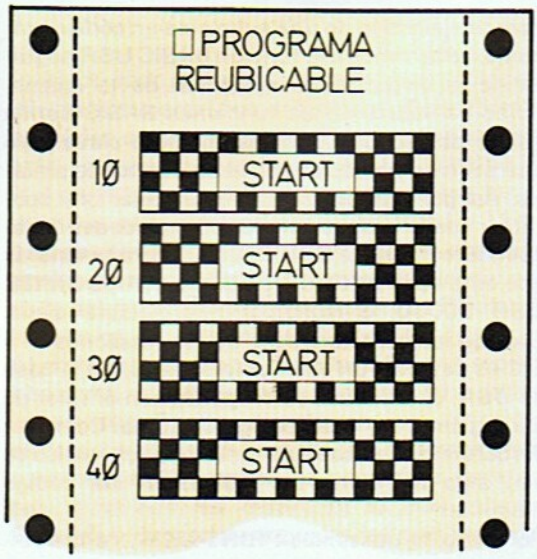
```
10 LET A=199
20 POKE 22528,A
```

El programa funciona correctamente cualquiera que sean los números de línea que se utilicen, respetando lógicamente dos normas evidentes: las líneas han de encontrarse en el mismo orden, y el programa se debe ejecutar desde el punto adecuado (en nuestro caso la primera instrucción). Como prueba de ello, tenemos el siguiente programa, que habrá de ejecutarse mediante **RUN 50** (o simplemente **RUN**).

```
50 LET A=199
98 POKE 22528,A
```

Al igual que para el BASIC los puntos de referencia son los números de instrucción, para el lenguaje máquina lo son las direcciones de memoria donde se encuentra. Vamos a cambiar de lugar nuestro programa en M/C (*machine code* o código máquina) del cuadrado intermitente: **POKE 31000,62: POKE 31001,199: POKE 31002,50: POKE 31003,0: POKE 31004,88: POKE 31005,201.**
Como recordaremos, este programa se ejecuta desde la primera dirección, que en este caso ya no es 30000, sino 31000. Efectivamente, con **RANDOMIZE USR 31000** seguirá funcionando a la perfección. Este programa es por tanto REUBICABLE también en código máquina. Sin embargo, esta propiedad no depende de la misión que cumpla el programa, sino de la manera en que es llevado a cabo. Así pues, podemos encontrar programas que cumplan un mismo objetivo en BASIC y en código máquina, y que sin embargo en un caso sean reubicables y en otro no. De todas formas, la reubicabilidad es un factor de comodidad muy a tener en cuenta, pero lamentablemente no es posible en todas las ocasiones.

Los programas que funcionan cualquiera que sea la referencia que se les otorgue, se denominan reubicables.



BITS

Por corto que sea un programa en lenguaje máquina, por ejemplo un mero código de retorno (201), siempre tarda en ejecutarse un tiempo mínimo, debido al trabajo del intérprete BASIC en ejecutar el **USR** BASIC, y manejar los parámetros de retorno según ha indicado la palabra clave que precede a la función.

*

Para hacernos una idea de la velocidad del lenguaje máquina, basta con saber que la instrucción más básica del Z-80 tarda en ejecutarse 4 estados (T). En el caso del Spectrum, **4T=4/3500000 seg.= 0.00000114285741 seg.**: y la más complicada, **23T = 23/3500000 seg. = 0.00000657142857**. Es decir, su velocidad de ejecución por código oscila entre menos de 2 microsegundos (millonésimas de segundo) y menos de 7 microsegundos.

En el siguiente ejemplo, veremos un programa BASIC no reubicable:

```
10 LET A=199
20 POKE 22528,A
30 GO TO 50
40 NEW
50 STOP
```

Este programa se ejecutará perfectamente desde la primera instrucción, pero sin embargo, sólo con hacer un pequeño cambio en los números de línea, puede que tenga resultados totalmente diferentes, aún siendo ejecutado desde el punto adecuado (la primera de sus líneas):

```
10 LET A=199
20 POKE 22528,A
30 GO TO 50
50 NEW
55 STOP
```

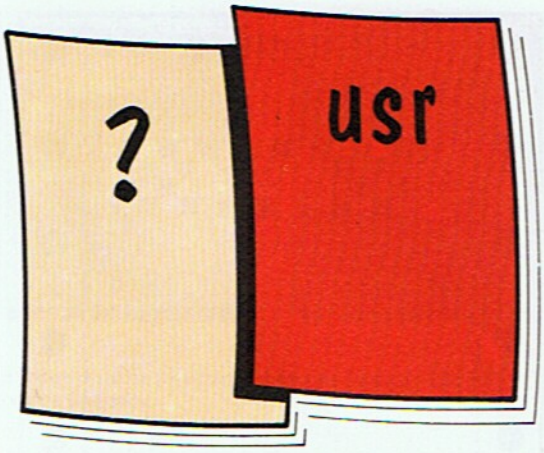
Esto mismo sucede con el código máquina. Aquellos programas que para funcionar correctamente, sin realizar en ellos ningún cambio, deben estar situados en determinado punto de la memoria se denominan NO REUBICABLES.

LA FUNCION USR

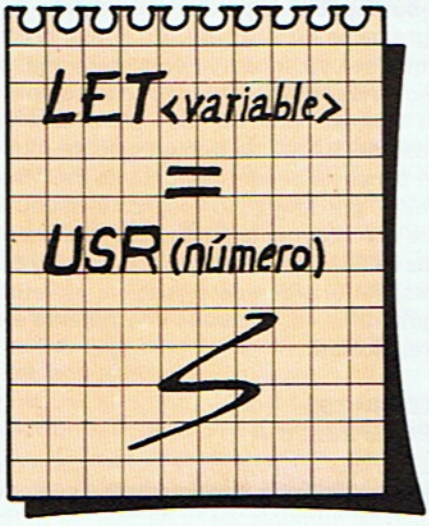
Para la ejecución de una subrutina en código máquina, recurrimos a la función BASIC **USR** seguida de la dirección de ejecución de la misma. Como ya sabemos, las funciones BASIC tienen que ir precededidas en una palabra clave para que sean admitidas por el comprobador de sintaxis. Así por ejemplo, si quieres ejecutar una subrutina desde la dirección 30000, no puedes introducir directamente **USR 30000**, sino que tendrás que emplear **PRINT USR 30000, RANDOMIZE USR 30000**, o cualquier otra palabra clave. Otra de las características de las funciones BASIC es devolver un valor determinado. En el caso de **USR**, el resultado corresponde con el valor de los registros B y C del código máquina. Como recordaremos del capítulo anterior, los registros son algo así como las «variables» del código máquina.

Por tanto, si utilizamos **PRINT USR**..., a la vuelta de la subrutina encontraremos un número escrito en la pantalla, puesto que la palabra clave utilizada como «trampolín» de **USR** ha sido **PRINT**. Para solucionar este problema, normalmente se utiliza la palabra clave **RANDOMIZE**, dado que no produce ningún efecto en la pantalla.

Pese a todo, este último sistema puede ocasionar ciertos problemas en programas que utilicen la generación de números aleatorios (**RND**). El método más eficaz para que la llamada al código máquina no provoque problemas, es el almacenamiento en una variable del resultado devuelto: **LET <variable> = USR** ... Por ejemplo, **LET A=USR 30000.**



La función USR *siempre debe ir precedida de una palabra clave.*



El sistema ideal para utilizar USR *es almacenar su resultado de retorno en una variable.*

Durante la ejecución de un programa en código máquina, puede ser que se produzca una detención con un extraño mensaje del Spectrum; esto será debido a que por un error en nuestra subrutina, la ejecución ha pasado a alguna zona de la ROM *(firmware)*.

*

La unidad de medida de tiempo de ejecución para el código máquina es el ESTADO (T). Este equivale a uno partido por la frecuencia de reloj del microprocesador. En el caso concreto del Z-80 que incorpora nuestro Spectrum, la frecuencia de reloj es de 3.500.000 Hz, es decir, **T=1/3500000 seg.**

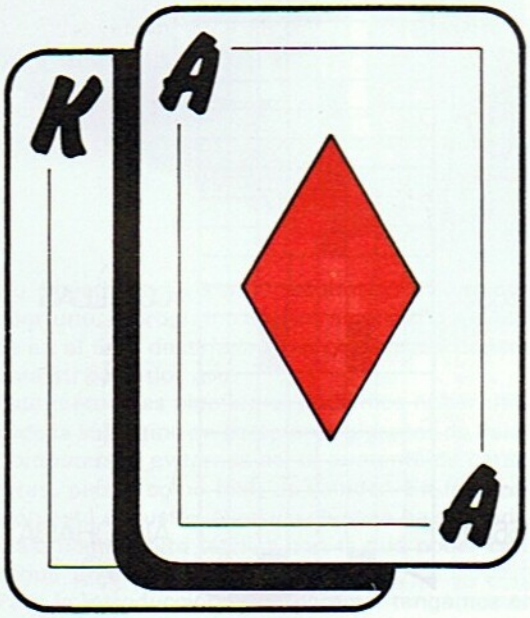
BLACK JACK

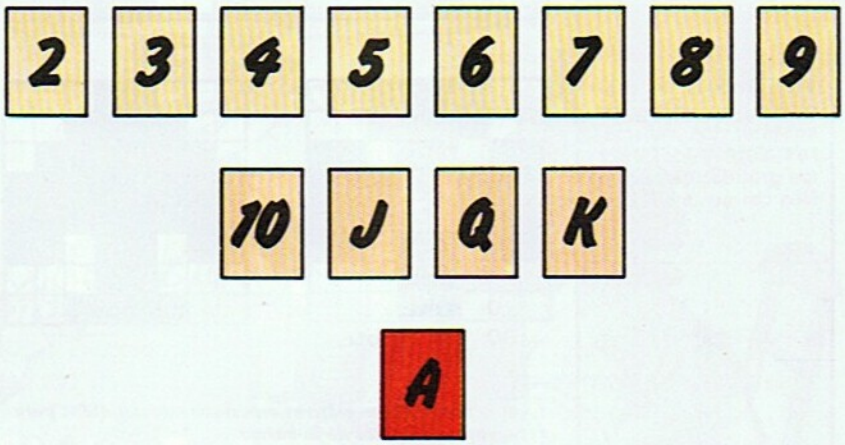
HEMOS creado para nuestros incondicionales lectores el RUN'S CASINO. Como acabamos de inaugurarlo, andamos escasos de croupieres que atiendan la totalidad de nuestras mesas; pero nuestro fiel amigo Spectrum se ha presentado voluntario para atender la mesa de Black Jack. Muchos de nosotros hemos oído hablar del Black Jack, pero desconocemos sus reglas de juego. Por ello, a continuación aparece una pequeña sinopsis de dichas normas, para que así lleguemos a ser unos empedernidos y expertos jugadores de Black Jack.

Antes de nada hemos de anticipar que las reglas de nuestro Black Jack no son exactamente las tradicionales del casino. Su mayor mejora consiste en que no se juega con pesetas, sino con una nueva moneda: el RUN DOLAR; un considerable ahorro para aquellos que no tienen buena fortuna con las cartas.

Por otra parte, aunque algunos desconozcamos el juego, su mecanismo no nos será del todo ajeno, puesto que es muy similar al popularísimo «siete y media».

La mejor jugada es el Black Jack, la cual se premia con el doble de lo apostado.





Los valores de los naipes corresponden con los números de las cartas, a excepción de las figuras, que tienen valor diez, y el as, el cual puede adoptar indistintamente los valores once o uno.

REGLAS DEL JUEGO

El Black Jack se juega con una baraja francesa (diamantes, corazones, tréboles y picas). El valor numérico de las cartas, es el impreso en las mismas, excepto en el as (A), el jack (J), la dama (Q) y el rey (K), cuyos valores respectivos son los siguientes: el as puede aumir los valores 1 u 11 según nos convenga, aunque en nuestro caso particular va a ser el Spectrum quien dé el valor que se le antoje dentro de los dos citados. El valor de las tres cartas restantes (J,Q,K) es 10.

Al comenzar el juego, el ordenador nos reparte una carta, y se despacha a sí mismo otra, ambas boca arriba. Aunque nuestros naipes aparezcan al descubierto en la pantalla, no debemos preocuparnos: el Spectrumn es honrado a carta cabal, y no nos las mirará ni lo más mínimo. El objetivo del juego es quedar los más próximos

BITS

La grabación del programa debe llevarse a cabo mediante el comando **SAVE "BLACKJACK"**. Si utilizamos el sistema de autoejecución, ésta ha de comenzar a partir de la linea 1 (**SAVE "BLACKJACK" LINE 1**).

¡!

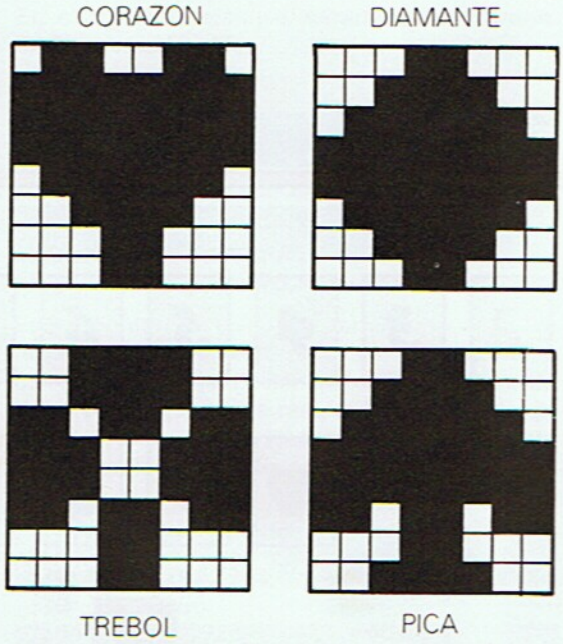
Al introducir el programa, no olvidemos sustituir los caracteres subrayados, por los gráficos de las teclas correspondientes.

*

En este programa, la duplicación de los caracteres se obtienen mediante una subrutina en BASIC, la cual accede a la dirección de la ROM donde se encuentrán almacenados los códigos que definen los caracteres del Spectrum.

*

El premio por obtener la jugada de Black Jack, es en nuestro juego, el doble de lo apostado.



Cuatro han sido los gráficos especialmente definidos para representar los palos de la baraja.

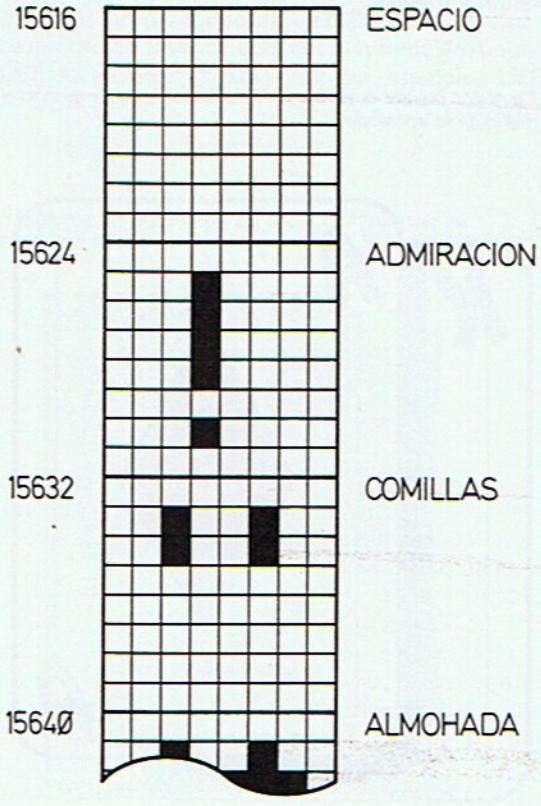
posible a veintiún puntos, pero sin sobrepasarlos, puesto que en ese momento quedamos descalificados. Para conseguirlos, a la vista de nuestras cartas, y de la que le vemos al ordenador, podremos pedir un naipe más, hasta que nos plantemos en determinada puntuación. El número máximo de cartas que podemos solicitar es cinco. Una vez llegados a este punto, será al Spectrum al que le toque jugar, aunque en su caso el abanico de posibilidades es mucho más restringido: tendrá que ir sumando puntos hasta un mínimo de 17, estando obligado a plantarse en el momento en que alcance o supere dicha cantidad. A continuación se procede a la comprobación de cual de los dos oponentes ha obtenido la mejor jugada. En este aspecto, el Spectrum se guarda dos bazas bastante importantes: la primera es que en caso de empate a puntos, siempre se declara vencedor el croupier, y la segunda, que si hemos sobrepasado la puntuación de veintiuno, habremos perdido la «mano», sin tener para nada en cuenta si a él tampoco le acompañó la suerte. Por último, existe una jugada que supera a todos; incluso a la de obtener veintiún puntos. Esta maravillosa combinación de cartas se denomina Black Jack, y consiste en obtener de entrada, es decir, con las dos primeras cartas, una puntuación de veintiuno (un as y una carta de valor diez). Así pues, podemos resumir las reglas del juego en los cinco puntos siguientes:

1. Se consigue Black Jack cuando con nuestras dos primeras cartas obtenemos veintiún puntos.
2. Si una vez pedidas dos cartas no hemos conseguido Black Jack, seguiremos sumando puntos, intentando acercarnos lo más posible al valor 21.
3. Podemos plantarnos en cualquier momento del juego, siempre que no hayamos sobrepasado los veintiún puntos, en cuyo caso ya no tendría sentido seguir la mano.
4. Si el Spectrum iguala nuestra puntuación habremos perdido esa baza, y en consecuencia lo que hubiésemos apostado.
5. Si la puntuación del Spectrum es 16 o menor, nuestro crupier personal tiene la obligación de pedir una nueva carta. Por el contrario, si la suma de los puntos obtenidos por el Spectrum es 17 o superior, se plantará, aunque con ello sepa que pierde la jugada.

La arquitectura del programa es completamente estructurada y sencilla de comprender, a excepción quizá de la subrutina que duplica el tamaño de los caracteres impresos en los naipes. Esta parte del programa se encuentra en la línea 8000, y a breves rasgos, funciona de la manera que a continuación explicamos.

La subrutina busca a partir de la dirección de memoria 15616 (ROM), el carácter elegido por el programa para su aumento. A partir de dicha dirección, se hayan almacenados todos los caracteres, tanto numéricos como alfabéticos, de nues-

A partir de la dirección de memoria 15616 se encuentra el área de generación de caracteres.



tro Spectrum. Una vez encontrado el carácter oportuno, el programa duplica su tamaño y lo ubica en el área destinada a los gráficos de usuario para su posterior uso.

Este método es algo lento. Podíamos haber utilizado la subrutina de caracteres gigantes de Psion Computers, y evitarnos así el aumento de caracteres, pero... como bien es sabido: «En la variación está el gusto». Nuestro objetivo ha sido además, ilustrar otra técnica con la que poder conseguir un efecto similar.

Para la introducción del programa, tengamos en cuenta que los caracteres subrayados corresponden a los gráficos de las teclas afectadas; así por ejemplo, A equivale al carácter gráfico de la A, es decir, el que se obtiene pulsando la tecla A cuando nos encontramos en el modo GRAPHICS (cursor G). Por otra parte, los caracteres doblemente subrayados (este hecho sólo se da con números), corresponden a los gráficos cambiados de los números afectados, los cuales se obtienen de forma similar a los anteriores, pero pulsando además **CAPS SHIFT.**

Para finalizar, resta advertir que es una buena

idea destacar los **REM** del programa escribiéndolos en vídeo inverso, de forma que sean más fácilmente localizables al efectuar un listado por pantalla. Los literales subrayados **INV.** entre corchetes, y **TRUE**, también entre corchetes, deben de interpretarse por tanto como **INVERSE VIDEO** y **TRUE VIDEO**, respectivamente; o lo que es lo mismo, **CAPS SHIFT + 4** y **CAPS SHIFT + 3**.

```
  10 REM ************************
  20 REM * J.M.MAYORAL SERRANO *
  30 REM ************************
  40 REM *   BLACKJACK  1985   *
  50 REM ************************
  60 LET DIREC=15360
  70 LET F$="": BRIGHT 0
  80 GO TO 140
  90 PRINT  INK 2; PAPER 7;AT P1-3,COL+1;"3333"
 100 FOR W=P1-2 TO P1+2
 110 PRINT  PAPER 7;AT W,COL+1;"    "
 120 NEXT W
 130 RETURN
 140 BORDER 4
 150 PAPER 4
 160 INK 9
 170 CLS
 180 REM [INV.] INIC. VARIABLES [TRUE]
 190 GO SUB 940: LET B$=F$
 200 LET VOC=RIG
 210 LET DIN=1000
 220 GO SUB 810
 230 LET IND=ABS SQR (VOC+4*RIG)+2
 240 LET IND=IND+2
 250 GO SUB 1530
 260 REM [INV.] FIN DEL JUEGO [TRUE]
 270 IF DIN<=0 THEN  PRINT  INK 9; PAPER 2; FLASH 1;A
T 10,3;"TE HAS QUEDADO SIN DINERO!";AT 19,2; PAPER 6;
"PULSA "C" PARA OTRA PARTIDA!": GO TO 1510
 280 REM [INV.] CONT. JUEGO [TRUE]
 290 PRINT  INK 9; PAPER 2;AT 10,2;"POSEES UN FONDO D
E ";DIN;" RUN$": PRINT AT 19,1; PAPER 6; INK 9;"CUANT
O DINERO QUIERES APOSTAR?"
 300 INPUT "RUN$= ";PASTA
 310 CLS
 320 IF PASTA>DIN THEN  GO TO 290
 330 IF PASTA<=0 THEN  GO TO 290
 340 IF PASTA-INT PASTA<>0 THEN  GO TO 290
 350 LET DIN=DIN-PASTA
 360 REM [INV.]  MENSAJES  [TRUE]
 370 PRINT  INK 9; PAPER 6;AT 0,0;"          TUS  CAR
TAS          "; PAPER 1;AT 11,0;"       CARTAS    SP
ECTRUM          "
 380 PRINT  INK 9;AT 9,1;"Total=";AT 20,1;"Total="
 390 REM [INV.]  CALCULOS  [TRUE]
 400 REM [INV.]  ALEATORIEDAD CARTAS [TRUE]
 410 FOR I=2 TO 1 STEP -1
 420 GO SUB 650
 430 NEXT I
 440 FOR I=1 TO 2
 450 LET VOC=U(I)=11
 460 IF I=1 THEN  PRINT  FLASH 1; PAPER 2; INK 7;AT 2
1,2;" QUIERES UNA CARTA MAS ? ": LET A$=INKEY$: GO TO
 480
 470 GO TO 520
 480 BEEP .05,40: LET A$=A$+" "
 490 IF A$(1)="N" OR A$(1)="n" THEN  GO SUB 1470: GO
TO 570
 500 IF A$(1)="S" OR A$(1)="s" THEN  RANDOMIZE : GO S
UB 1470: GO TO 520
 510 GO TO 460
 520 GO SUB 650
 530 IF U(I)>10+(RIG+1)*11 THEN  LET I=2: GO TO 570
 540 IF U(I+RIG+2)=-20+(17*RIG)+25 THEN  GO TO 570
 550 IF I=VOC*RIG+2 AND U(I)>(RIG+4)*4 THEN  GO TO 57
0
 560 GO TO 460
 570 NEXT I
 580 IF U(RIG+1)<=21 AND U(RIG+3)=5 AND (U(RIG+2)<>21
 OR U(RIG+4)<>((RIG+1)*2)) THEN  GO TO 620
 590 REM [INV.]  BLACKJACK  [TRUE]
 600 IF U(RIG+1)=21 AND U(RIG+3)=(RIG+1)*2 AND (U((RI
G+1)*2<>21 OR U(RIG+4)<>2)) THEN  LET PASTA=PASTA*1.5
: PRINT  FLASH 1;AT 5,18;" BLACKJACK ": GO TO 620
 610 IF (U((RIG+1)*2)<=21 AND U(2)>=U(RIG+1)) OR U(RI
G+1)>21 THEN  LET PASTA=0
 620 LET DIN1=DIN
 630 LET DIN=DIN+(RIG+2)*PASTA
 640 GO TO 220
 650 LET QUO=I+2
 660 LET U(QUO)=U(QUO)+1
 670 LET C=INT (RND*13)+2
 680 IF C=VOC*RIG+11 THEN  LET VOC=VOC+1
 690 LET U(QUO-2)=U(QUO-2)+C*(C<12)+10*(C>11)
 700 IF RIG=2 THEN  GO SUB 1180
 710 IF RIG*VOC>RIG THEN  GO TO 650
 720 IF U(I)<20+RIG+2 OR VOC=RIG THEN  GO TO 750
 730 LET VOC=VOC-1
 740 LET U(I)=U(I)-10
 750 GO SUB 1010
 760 IF NOT RIG THEN  GO SUB 1180
 770 LET FILA=RIG-2+(I*11): LET COLUM=9
 780 PRINT  INK 0;AT FILA,COLUM-1;U(I)
 790 RETURN
 800 REM [INV.] GRAFICOS USUARIO [TRUE]
 810 DATA 0,78,209,81,81,81,78,0
 820 DATA (RIG+51)*2,255,255,255,126,60,24,24
 830 DATA 24,60,126,255,255,126,60,24
 840 DATA 24,60,126,255,255,219,24,60
 850 DATA RIG+6*10,RIG+6*10,219,231,231,219,24,24
 860 DATA 78,50,51,52,53,54,55
 870 DATA 56,57,84,65,74,81,75
 880 RESTORE 810
 890 FOR K=USR "E" TO USR "E"+39
 900 READ X
 910 POKE K,X
 920 NEXT K
 930 RETURN
 940 RESTORE 860: FOR B=1 TO 14
 950 READ A: LET F$=F$+CHR$ A
 960 NEXT B
 970 LET RIG=0
 980 LET POS=RIG+4
 990 RETURN
1000 REM [INV.] DIBUJO CARTAS [TRUE]
1010 LET P=11*I+(POS-11)
1020 LET COL=6*U(I+POS-2)-5
1030 LET P1=P
1040 LET Q1=COL
1050 INK 2: PAPER 4
1060 PRINT AT P-3,COL;"533335"
1070 FOR K=1 TO 5
1080 PRINT AT P-3+K,COL;"5    5"
1090 NEXT K
1100 PRINT AT P+3,COL;"133332"
1110 INK 0
1120 GO SUB 90
1130 LET COLA=INT (RND*POS)+1
1140 INK 2*(COLA<3)
1150 PRINT  PAPER 7;AT P-2,COL+3;CHR$ (148+COLA)
1160 RETURN
1170 REM [INV.]  SBR DUPLIC. CARAC. [TRUE]
1180 LET X=USR "A"
1190 FOR K=0 TO 31
1200 POKE X+K,0
1210 NEXT K
1220 BRIGHT 0
1230 PAPER 7
1240 LET S=POS*2*CODE B$(C)+DIREC
1250 IF B$(C)=CHR$ 84 THEN  LET S=USR "E"
1260 FOR J=0 TO 7
1270 LET V=PEEK (S+J)
1280 LET C=0
1290 LET D=64*POS
1300 FOR K=8 TO 1 STEP -1
1310 LET D=D/2
1320 IF V>=D THEN  LET C=C+3*D*D: LET V=V-D
1330 NEXT K
1340 LET H=INT (C/256)
1350 LET L=C-256*H
1360 POKE X,H
1370 POKE X+1,H
1380 POKE X+8,L
1390 POKE X+9,L
1400 LET X=X+2+8*(J=3)
1410 NEXT J
1420 PRINT  BRIGHT 1;AT P,COL+2;"AB"
1430 PRINT  BRIGHT 1;AT P+1,COL+2;"CD"
1440 PAPER 4
1450 BRIGHT 0:
1460 RETURN
1470 REM [INV.]  BORRADO MENSAJE [TRUE]
1480 PRINT  PAPER 4;AT 21,0;"
                                    "
1490 RETURN
1500 REM [INV.] OTRA JUGADA ? [TRUE]
1510 IF INKEY$="C" OR INKEY$="c" THEN  RUN
1520 GO TO 1510
1530 DIM U(IND): RETURN
```

ATRIBUTOS DE IMPRESION

Si tenemos en cuenta el hecho de que los colores pueden adoptar el brillo normal y el especial, podemos decir que contamos con una paleta de dieciséis colores para trabajar, siempre dentro del área imprimible de la pantalla, puesto que para el marco sólo es posible el empleo de los 8 básicos.

Esto nos sugiere la idea de crear una tabla de color, compuesta por el código de éste y su brillo, asociado a un nombre de color concreto, tal como se detalla en la tabla adjunta.

Esta tabla es igualmente aplicable a la sentencia **PAPER** y a la **INK**, puesto que se trata de un atributo de la posición de pantalla. De esta forma, cuando es necesario imprimir en un lugar concreto de la misma, el Sistema analiza los colores asignados a la posición, y utiliza para los puntos iluminados el que indica **INK** y para los no iluminados el de **PAPER**, quedando los dos afectados por el brillo de la posición (normal o suplementario).

Sabido esto, es fácil comprender que **FLASH** se limita a alterar el contenido de los atributos de tinta y color, de forma intermitente, consiguiendo el efecto parpadeante en la posición o posiciones de pantalla afectadas.

TABLA DE COLOR

CODIGO	NOMBRE	COLOR SIN BRILLO	COLOR CON BRILLO
0	Black	Negro	Negro resaltado
1	Blue	Azul marino	Azul marino claro
2	Red	Rojo oscuro	Rojo claro
3	Magenta	Rosa oscuro	Rosa claro
4	Green	Verde oscuro	Verde claro
5	Cyan	Azul celeste	Azul celeste claro
6	Yellow	Amarillo oscuro	Amarillo claro
7	White	Gris claro	Blanco

CODIGOS DE COLOR ESPECIALES

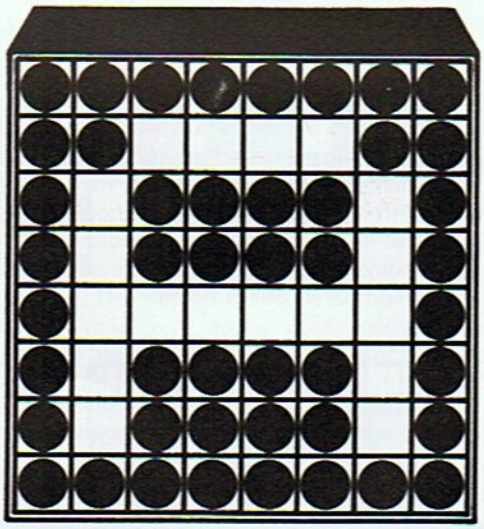
Los códigos especiales son los valores 8 y 9 de los parámetros de color. Al ser números mayores que siete, es evidente que no pasan a formar parte directa del atributo de color de la posición, sino que señalan al Sistema determinada forma de interpretación.

El código 8 indica color TRANSPARENTE. Se trata, pues, de un código de color neutro, que no altera en absoluto el elegido anteriormente para la posición, pudiendo emplearse con **PAPER**, **INK**, **BRIGHT** y **FLASH**.

En principio, podría parecernos que el uso de este código de control es lo mismo que no emplear ninguna sentencia de color. Realmente es así, aunque en determinadas ocasiones simplifica la programación.

Veamos un ejemplo. Supongamos que al comien-

En la pantalla del ordenador, los puntos encendidos se representan en el color de primer término (INK) *y los apagados en el de fondo* (PAPER).



PAPER: Ø
INK: 2

¡!

El juego completo de colores del Spectrum, combinando las dos intensidades de brillo, lo integran dieciséis tonalidades diferentes.

*

Los parámetros utilizables para los códigos de color van desde el cero hasta el siete, más dos valores especiales: el ocho y el nueve, correspondientes a los efectos de transparencia y contraste, respectivamente.

*

El código de transparencia no altera el color de las posiciones a las cuales afecta.

*

El código de contraste equivale a siete (blanco), o cero (negro), según las necesidades tendentes a obtener una máxima legibilidad de los caracteres escritos.

zo de un programa, hemos asignado a la pantalla un determinado color de fondo; a continuación, unas instrucciones más adelante, tras haber utilizado reiteradas veces sentencias **INK** y **PAPER**, condicionadas mediante **IF** o por cualquier otro sistema, no tenemos conocimiento del color de fondo existente actualmente en la pantalla. Si quisiéramos ahora escribir un carácter en dicho color de fondo, nos sería imposible, dado que no sabríamos qué parámetro especificar para **PAPER**. Pues bien, indicando el código 8, le encargamos al ordenador que sea él el que averigüe el color actual de la pantalla, escribiendo el carácter sin modificar éste.

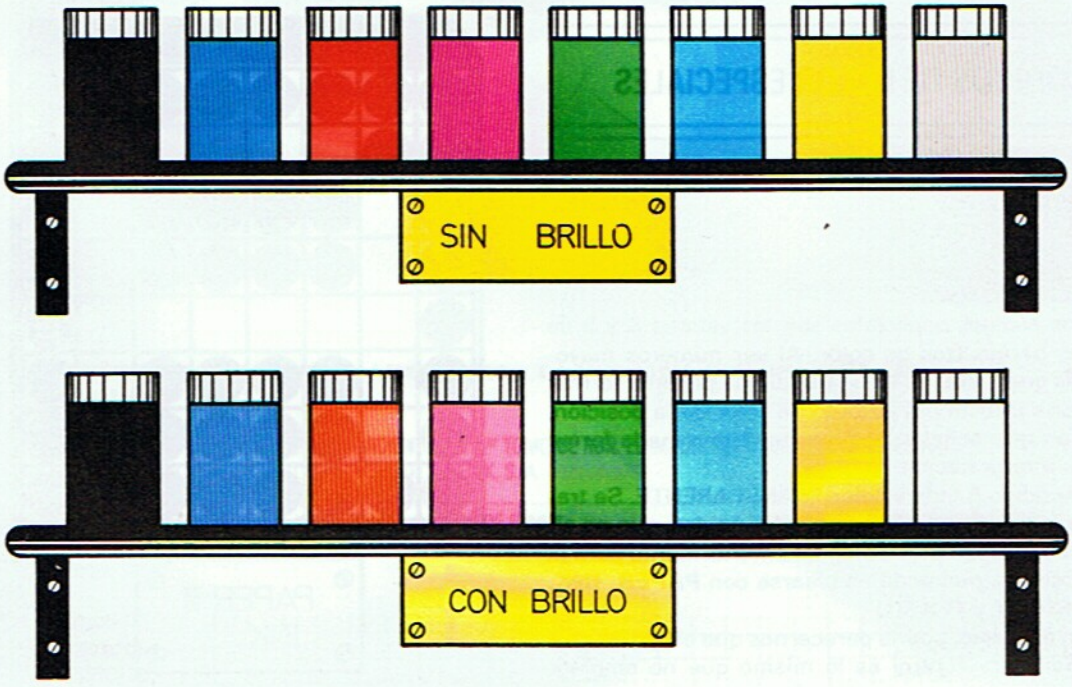
El código 9 corresponde al color de CONTRASTE, siendo únicamente aplicable a los atributos **PAPER** e **INK**. Este selecciona, cuando se aplica, el color adecuado del atributo que se desea modificar en función del valor del otro, de forma que se obtenga la máxima visibilidad.

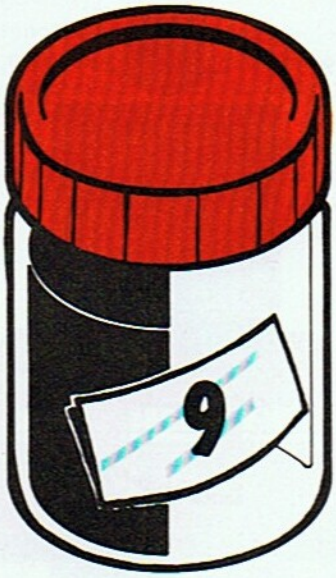
Con esto conseguimos que el Spectrum asigne, **INK 0** (negro) para los **PAPER** de tonalidad clara e **INK 7** (blanco) para los de tonalidad oscura. Para hacer esto, el ordenador sabe que los códigos de color del 0 al 3 corresponden a los tonos oscuros, y los comprendidos entre el 4 y el 7 a los claros.

De manera similar, al emplear **PAPER 9**, el Spectrum asigna los códigos de fondo del papel de forma que presenten un contraste con la tinta existente en esa posición, es decir, fondo negro para las tintas claras, y blanco para las oscuras. La utilización del código de control 9, garantiza que los caracteres que escribimos sean siempre legibles. No obstante, conviene saber que este código emplea tan solo los tonos blanco y negro, por lo que si nos referimos a los dos atributos de determinada posición con este mismo código, es

El código de color ocho simula una tinta transparente.

Manejando convenientemente la posibilidad de brillo suplementario, nuestro Spectrum dispone de un total de dieciséis colores.





El código de color nueve encuentra el punto de mayor contraste, utilizando para ello los colores blanco o negro, según convenga.

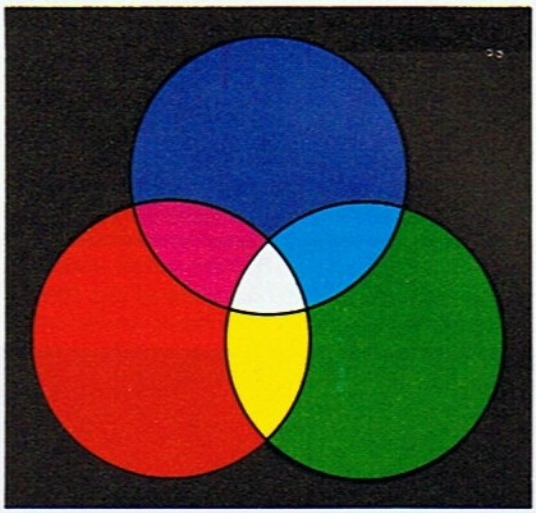
decir, asignando antes de escribir un carácter tanto **PAPER 9** como **INK 9**, obtendremos negro sobre blanco (estado al conectar el ordenador), o blanco sobre negro (vídeo invertido).
Veremos más claras estas combinaciones en el programa de ejemplo titulado EFECTOS CODIGO 9.

```
  10 REM - EFECTOS CODIGO 9
  20 INK 9
  30 FOR I=0 TO 7
  40 PAPER I
  50 FOR J=0 TO 1
  60 BRIGHT J
  70 PRINT "PAPER ";I;" ";
  80 FOR K=0 TO 23: PRINT "X";: NEXT K
  90 NEXT J: NEXT I
 100 BRIGHT 0
 110 PRINT ´´´TAB 4;"COMBINACIONES CON I
NK 9"
 120 PAUSE 500: CLS
 130 PAPER 9
 140 FOR I=0 TO 7
 150 INK I
 160 FOR J=0 TO 1
 170 BRIGHT J
 180 PRINT "INK ";I;" ";
 190 FOR K=0 TO 25: PRINT "X";: NEXT K
 200 NEXT J: NEXT I
 210 PAPER 7: INK 0: BRIGHT 0
 220 PRINT ´´´TAB 3;"COMBINACIONES CON P
APER 9"
```

OBTENCION DE LOS COLORES BASICOS

Ya hemos dicho que los colores básicos son los ocho que se corresponden con las teclas del 0 al 7, en la fila superior del teclado. En cualquier caso, estos colores básicos no son más que combinación de tres fundamentales, sobre los cuales se desarrolla la generación del color en los receptores de TV.
Estos tres colores realmente fundamentales son el azul, el verde y el rojo. Por combinación de ellos, se obtienen todos los demás.
Si somos un poco observadores, encontraremos unas relaciones muy interesantes entre los códigos y los colores. La combinación de azul (código 1), con el rojo (código 2), nos da el color magenta (código 3). Del mismo modo, la combinación del color rojo (código 2), con el verde (código 4), nos da el color amarillo (código 6). Y así sucesivamente, los códigos van coincidiendo con la suma de los colores que los componen.
Por supuesto, esta relación de suma no es una pura casualidad. Como norma general, podemos decir que los 5 colores básicos del Spectrum, aparte del azul, el verde y el rojo, se corresponden con las combinaciones de las diferentes sumas de los fundamentales entre ellos.
Para hacernos una idea más gráfica de todo esto, podemos imaginar un escenario oscuro (obtenido por ausencia de todos los colores), sobre el cual

La generación de los ocho colores básicos del Spectrum se fundamenta en la combinación de los tres primarios: rojo, azul y verde.



Con los atributos **INK, PAPER, BRIGHT** y **FLASH**, se pueden emplear los parámetros **1** y **0**, para su conexión y desconexión respectivamente, así como el código **8**, utilizado para el efecto de transparencia.

*

La zona de la memoria en la cual el ordenador almacena los atributos correspondientes a cada carácter de la pantalla, se denomina ZONA DE ATRIBUTOS.

*

En la ZONA DE ATRIBUTOS se hayan representados los estados de **INK, PAPER, BRIGHT** y **FLASH** de cada posición de carácter de la pantalla.

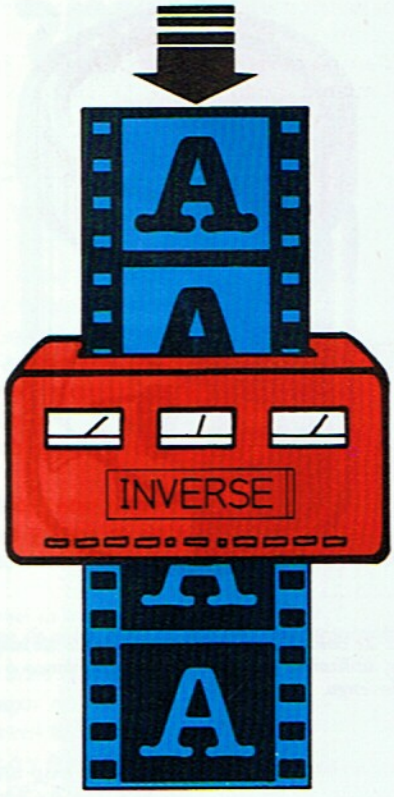
*

La instrucción **OVER 1** nos da acceso a la superposición de caracteres.

dirigiremos tres haces de luz con los colores azul, verde y rojo. El resultado de este curioso experimento sería más o menos el que se obtiene mediante el programa titulado OBTENCION DE LOS COLORES BASICOS.

```
  10 REM - OBTENCION COLORES BASICOS
  20 BORDER 0: PAPER 0: INK 7: CLS
  30 PRINT "OBTENCION DE LOS COLORES BAS
ICOS"''''
  40 FOR K=130 TO 150 STEP 10: GO SUB 70
: NEXT K
  50 PRINT AT 4,8;"2";AT 10,8;"3";AT 10,
14;"7";AT 10,20;"6";AT 16,14;"5";AT 19,2
;"1";AT 19,29;"4"
  60 STOP
  70 FOR I=0 TO 5
  80 RESTORE K: PRINT " ";
  90 FOR J=0 TO 4: READ X: PRINT  INK X;
"       ";: NEXT J
 100 PRINT
 110 NEXT I
 120 RETURN
 130 DATA 0,2,2,2,0
 140 DATA 1,3,7,6,4
 150 DATA 1,1,5,4,4
```

De esta forma se generan los 8 colores básicos. Los tres primeros por ser primarios (esta es la denominación técnica que reciben los mencionados tres colores fundamentales), tres más como combinación directa de ellos tomados dos a dos, y uno más, el blanco, por combinación de los tres fundamentales.



El efecto de INVERSE 1 *en la pantalla es similar a la creación de un negativo fotográfico.*

La zona de la memoria donde el ordenador almacena la disposición de los puntos encendidos y apagados en la pantalla, al margen de los atributos, se denomina ARCHIVO DE IMAGEN.

Cuando un mismo carácter se escribe en **OVER 1** dos veces consecutivas sobre una misma posición, se recupera el carácter primitivo sobre el cual se superpuso.

LA SENTENCIA INVERSE

Cada carácter representado en la pantalla, se representa dentro de un cuadrado (posición de carácter) configurado por 64 puntos, distribuidos en 8 filas de 8 columnas. Dentro de esta estructura reticular, los puntos iluminados adoptan el color del atributo **INK**, y los no iluminaods el de **PAPER**.

Hemos visto también, que el efecto de **FLASH** es intercambiar los valores de **INK** y **PAPER**, de forma intermitente, para conseguir una intermitencia.

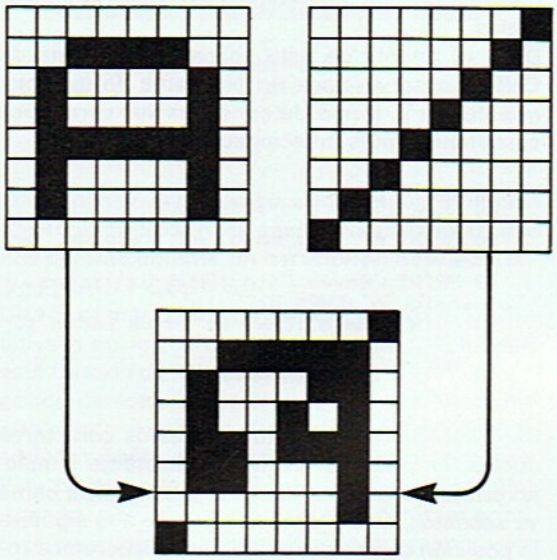
INVERSE hace algo muy parecido, aunque de una forma permanente: intercambia los valores de **INK** y **PAPER** de determinada zona de la pantalla, para obtener lo que se denomina comúnmente como vídeo inverso; similar a lo que vemos en un negativo fotográfico.

INVERSE tiene también una paralelismo evidente en su utilización con **BRIGHT** y **FLASH**, puesto que existe también un **INVERSE 1** y un **INVERSE 0**. **INVERSE 1** marca el inicio de la actividad de esta función de impresión, e **INVERSE 0** determina el final.

La combinación de FLASH *e* INVERSE *nos permite mostrar de una manera muy original los caracteres en la pantalla.*

En cualquier caso, conviene que nos fijemos en que esta función de impresión no afecta a los atributos de una posición determinada de la pantalla, del mismo modo en que lo hacen las opciones anteriormente descritas.
En este caso, se trata de una opción que califica a la totalidad de la impresión, pero sin dejar huella en la zona de atributos de las posiciones de pantalla.

La función OVER *no efectúa exactamente una simple sobreimpresión.*

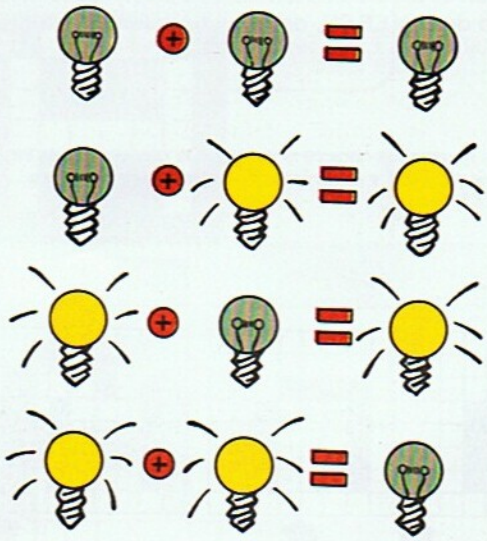


Aunque más adelante trataremos en profundidad este tema, haremos ahora una pequeña aclaración al respecto. Una zona de la memoria del ordenador, denominada ZONA DE ATRIBUTOS, es utilizada por éste para almacenar los códigos de color que corresponden a cada una de las posiciones de la pantalla.
Cuando ejecutamos una sentencia **FLASH, BRIGHT, INK** o **PAPER,** los estados que indican quedan reflejados en dicho área de la memoria. Sin embargo, al ejecutar una instrucción del tipo **INVERSE 1,** el ordenador se limita a trasponer directamente los datos de **INK** y **PAPER** en las siguientes impresiones, pero no reflejando en la zona de atributos si esto se debió a una sentencia **INVERSE,** sino simplemente como si el programador hubiera cambiado los códigos de color mediante **INK** y **PAPER.**
Puesto que la utilización de **INVERSE** es bastante frecuente, mostraremos un ejemplo de ello. El siguiente programa imprime el mensaje que le indiquemos, carácter a carácter, centrado en la pantalla, haciendo uso de la conexión y desconexión de la función de impresión **INVERSE** para producir un resalte más original que el de **FLASH:**

La sentencia OVER *establece la posibilidad de sobreescritura.*

```
 10 REM - INVERSE / FLASH - J.M. LOPEZ MARTINEZ
 20 INPUT "Titulo:"; LINE X$: IF X$="" THEN STOP
 30 LET X=LEN X$: IF X$>32 THEN GO TO 20
 40 CLS : FLASH 1
 50 FOR I=1 TO X
 60 PRINT AT 11,(32-X)/2+I-1; INVERSE I/2-INT (I/2);X$(I)
 70 NEXT I
 80 FLASH 0: GO TO 20
```

OVER *ejecuta una operación lógica XOR (o exclusivo, que se representa mediante un signo suma inscrito en un círculo), entre los puntos de los caracteres a representar.*

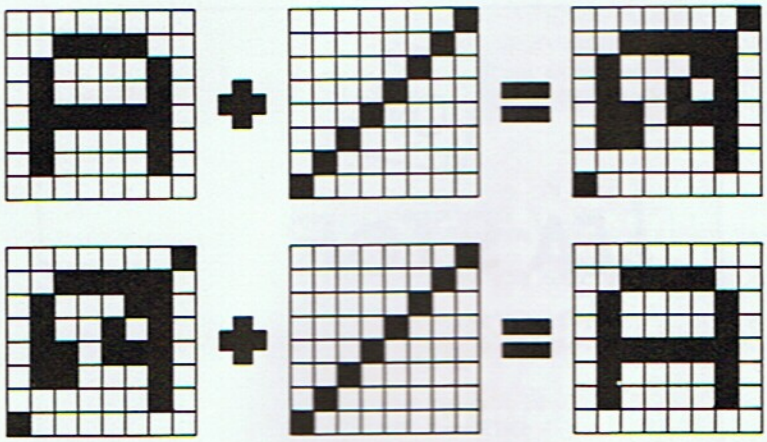


¡!

Las sentencias de atribución de color, combinadas con la sentencia **PRINT** en forma de función, tienen un valor temporal, dotado de mayor prioridad que el permanente.

*

Las sentencias de atribución de color utilizadas como palabras clave, tienen un valor permanente, empleado para todas las impresiones sucesivas, siempre y cuando éstas no conlleven atribuciones temporales.



Una propiedad de OVER *es la vuelta al carácter original con sólo escribir de nuevo el superpuesto.*

LA SENTENCIA OVER

Del mismo modo que **INVERSE** nos da paso a la posibilidad de impresión en vídeo inverso, **OVER** nos permite acceder a la sobreimpresión de caracteres.

OVER tiene también conexión y desconexión de la forma explicada anteriormente, es decir, por medio de **OVER 1** y **OVER 0**, respectivamente. Asimismo, esta sentencia no tiene influencia permanente sobre la configuración de los atributos de pantalla de las zonas a las que afecta (presencia en el área de atributos). Se trata, como en el caso de **INVERSE**, de una posibilidad de impresión externa, no controlada por los atributos. Cuando escribimos normalmente sobre la pantalla, cada nuevo carácter que coincide en una posición ocupada ya por otro, reemplaza al antiguo. Este hecho no debe importarnos normalmente, pero nos impide la sobreimpresión, es decir, la posibilidad de acceder a caracteres especiales como composición de dos o más simples.

En el caso de los programadores hispanoparlantes, esta facultad de impresión es de especial importancia, dado que nos permite obtener caracteres propios del Castellano, como la eñe **(ñ)** o las vocales acentuadas, que en un principio nos están vedados a los usuarios del Spectrum, puesto que no son caracteres disponibles en el alfabeto inglés.

Dado el interés de esta aplicación concreta de **OVER**, observaremos un programa de ejemplo, que ilustra la forma de conseguir los caracteres castellanos antes mencionados.

```
10 REM - OVER
20 OVER 1
30 PRINT AT 9,5;"Bajo  el  reinado  de
 los"''
40 PRINT "Reyes Cato";CHR$ 8;"'licos,
se consiguio";CHR$ 8;"' la"''
50 PRINT "unio";CHR$ 8;"'n de Espan";C
HR$ 8;"~a y Portugal."
60 OVER 0
```

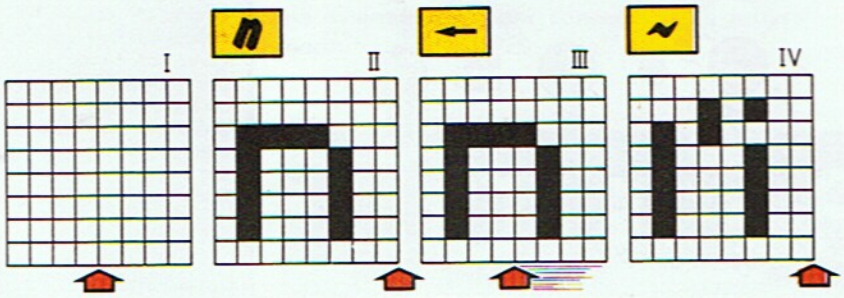
Para acceder a la escritura de estos caracteres dobles, empleamos un método bastante simple: en primer lugar, escribimos la letra básica; como ya sabemos, ésto produce el avance a la siguiente posición de escritura. Así pues, deberemos retroceder, para volver a situarnos sobre la letra básica, y aprovechándonos de **OVER**, escribir sobre ella, sin borrarla, el signo complementario deseado.

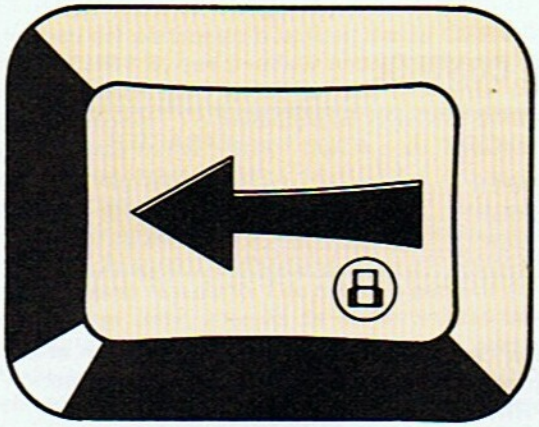
Así pues, obtendremos la eñe **(ñ)**, escribiendo una ene **(n)**, retrocediendo y escribiendo la línea superior que la caracteriza **(ñ)**. De modo similar, podremos obtener las vocales acentuadas, mediante la impresión previa de la vocal, el retroceso de un espacio y la sobreimpresión de un carácter de acentuación (ó), obtenido mediante **SYMBOL SHIFT + 7**.

De todo este proceso, el único elemento que todavía nos es ajeno es el sistema cómodo para retroceder el paso dado durante la primera impresión. Hace poco que acometimos el estudio del código A.S.C.I.I. de caracteres, y en él se puso de manifiesto que los 32 primeros caracteres (del 0 al 31), carecían de una representación en la pantalla. Estos son los denominados códigos de control, que no representan ningún carácter, sino que efectúan una determinada función.

Uno de estos códigos de control, concretamente el ocho, es el que realiza la función de retroceder una posición de carácter el puntero de escritura, se encuentre donde se encuentre actualmente. En lo referente a su obtención, y puesto

Para formar los caracteres castellanos nos apoyamos en la función OVER *y el carácter de control para retroceso* (CHR$ 8).





El carácter de control 8 (CHR$ 8) *produce el retroceso del puntero de impresión en una posición, se encuentre donde se encuentre.*

que conocemos su código, utilizaremos la función **CHR$**, estudiada hace apenas unas páginas, que nos permite obtener un carácter en base a su código **(PRINT CHR$ 8)**.

Una cosa más debemos conocer sobre **OVER**, relativa a su forma de actuar. Cuando se emplea esta función de impresión, se sobreimprime el segundo carácter sobre el primero, pero de forma que donde coinciden dos puntos iluminados, se sitúa un punto sin iluminar (del color del papel). Este hecho, que en principio puede parecer algo extraño, nos permite recuperar el carácter primitivo, repitiendo la operación de sobreimpresión con **OVER 1**. Como prueba de ello, podemos ver lo que sucede al ejecutar esta línea de programa:

10 CLS: OVER 1: PRINT "0";CHR$ 8;: PAUSE 250: PRINT "8": GO TO 10

Este resultado corresponde con el de una función lógica, denominada **XOR**, que será estudiada junto con otras funciones de este tipo en las próximas páginas dedicadas a la sección TU SPECTRUM.

ATRIBUTOS TEMPORALES

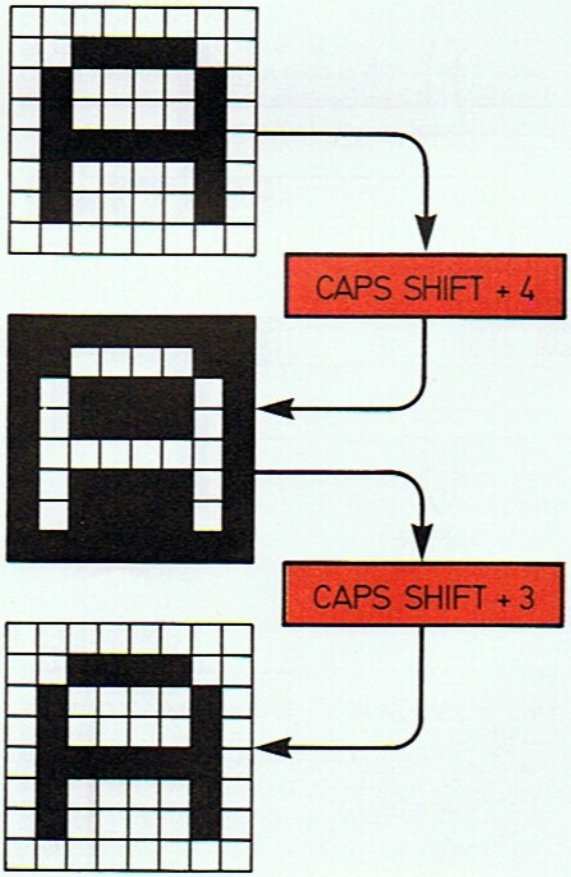
Hasta ahora hemos visto que los atributos afectan de forma permanente a las impresiones sucesivas, hasta que no se recibe una nueva instrucción para alterar este estado de cosas.

En ciertas ocasiones, es muy interesante disponer de una asignación temporal de los colores. Por ejemplo, podemos estar escribiendo habitualmente en blanco sobre azul, e interesarnos destacar las palabras iniciales de los párrafos en amarillo. Estar cambiando de color cada vez que se escribe el comienzo de párrafo, la primera vez para asignar el amarillo, y a continuación para restablecer el blanco habitual, puede resultar ciertamente engorroso. Para superar este problema, el Spectrum dispone de un sistema de asignación temporal de los atributos.

Este consiste en utilizar las palabras BASIC para control de atributos, en conjunción con una sentencia **PRINT.** Así por ejemplo, si nosotros escribimos: **INK 4: PRINT "A"**, la letra **A** será escrita en verde, como pretendíamos, pero también todos los caracteres que se escriban desde ese momento, hasta la ejecución de otra sentencia **INK**, se representarán en el mismo color.

En vez de esto, podemos sustituir la instrucción del ejemplo por **PRINT INK 4;"A"**. En un principio, el efecto parece idéntico, pero pronto comprobaremos que no es así, puesto que los carac-

Mediante las teclas 4 y 3, en combinación con CAPS SHIFT, *podemos acceder a los códigos de control de* INVERSE.



BITS

El estado de **OVER** o **INVERSE** no es representado en la zona de atributos, sino que afecta directamente al archivo de imagen (en el primer caso), y a los colores de **INK** y **PAPER** simultáneamente, en el segundo.

*

Los televisores obtienen cualquier color en base a la combinación de los tres fundamentales: rojo, azul y verde.

*

Los códigos que reciben los diversos colores básicos en el Spectrum, corresponden a la suma de los fundamentales mediante los cuales se obtienen.

*

La instrucción **INVERSE 1** permite trasponer los colores de fondo y primer término de cualquier carácter.

(Cuadro 2)

CARACTERES DE CONTROL

FUNCION	CARACTER	FUNCION	CARACTER
INK	CHR$ 16	PAPER	CHR$ 17
FLASH	CHR$ 18	BRIGHT	CHR$ 19
INVERSE	CHR$ 20	OVER	CHR$ 21

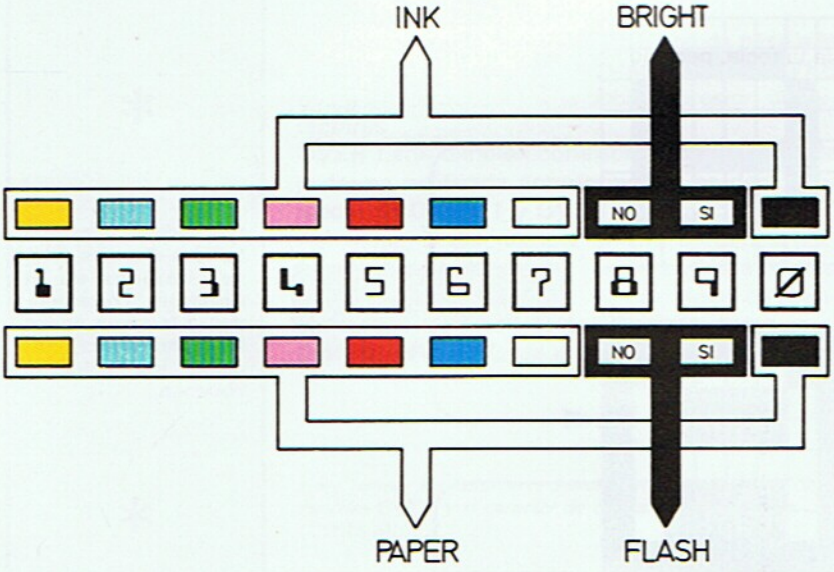
teres sucesivos serán escritos en el color inicial (el negro, al encender el ordenador), y no en verde.

Este mismo resultado puede conseguirse con cualquiera de los otros atributos: **PAPER, BRIGHT, FLASH, INVERSE** y **OVER,** seguidos de los parámetros habituales, incluyendo los especiales para transparencia y contraste.

Así pues, las palabras BASIC para el control de atributos, se pueden utilizar tanto en forma de palabras clave BASIC como de funciones. En el primer caso, su efecto será permanente; en el segundo, cesará cuando termine la instrucción **PRINT** en que se incluyen.

También es posible incluir funciones de color en varios puntos de una instrucción **PRINT,** con lo cual el efecto de éstas permanecerá durante todas las impresiones realizadas hasta el próximo cambio de atributo temporal. Para comprobarlo, ejecutemos la siguiente instrucción: **PRINT INK 4;"A"; TAB 15;"B"; AT 10,0;"C"; INK 2;"D": PRINT "E".**

Mediante la fila superior del teclado, es posible obtener caracteres de control del color.



LOS CARACTERES DE CONTROL DEL COLOR

Además de esta forma de incluir colores temporales, nuestro ordenador dispone de un par de sistemas más para cumplir el mismo objetivo, ambos basados en los códigos de control del color.

El primero de estos métodos consiste en sustituir la función de asignación de atributos por su carácter correspondiente en el código A.S.C.I.I., que lógicamente se encuentra en la zona de caracteres de control. Así pues, podremos sustituir **INK** por **CHR$ 16, PAPER** por **CHR$ 17 FLASH** por **CHR$ 18, BRIGHT** por **CHR$ 19, INVERSE** por **CHR$ 20** y **OVER** por **CHR$ 21.**

FUNCIONES DE IMPRESION

FUNCION	CARACTERES CONTROL	FUNCION	CARACTERES CONTROL
BRIGHT 0	CHR$ 19+CHR$ 0	BRIGHT 1	CHR$ 19+CHR$ 1
FLASH 0	CHR$ 18+CHR$ 0	FLASH 1	CHR$ 18+CHR$ 1
INVERSE 0	CHR$ 20+CHR$ 0	INVERSE 1	CHR$ 20+CHR$ 1

(Cuadro 3)

El código habrá de ir seguido por un carácter indicativo del parámetro a utilizar: desde **CHR$ 0** hasta **CHR$ 9**, que puede ir ensamblado con el atributo correspondiente, ya sea mediante un punto y coma (;) o mediante un operador suma (+). De esta forma, **PRINT INK 4...** es lo mismo que escribir **PRINT CHR$ 16;CHR$ 4...**, o bien **PRINT CHR$ 16+CHR$ 4...**

Existe aún una última posibilidad para incluir caracteres de control temporales: esta vez directamente desde el teclado. Ello nos va a permitir desarrollar el color, además de en las salidas impresas del programa, cosa que ya hemos hecho hasta ahora por el sistema convencional, dentro en las propias líneas de programa, para resaltarlas de un modo especial al efectuar un listado, realizar comentarios etc...

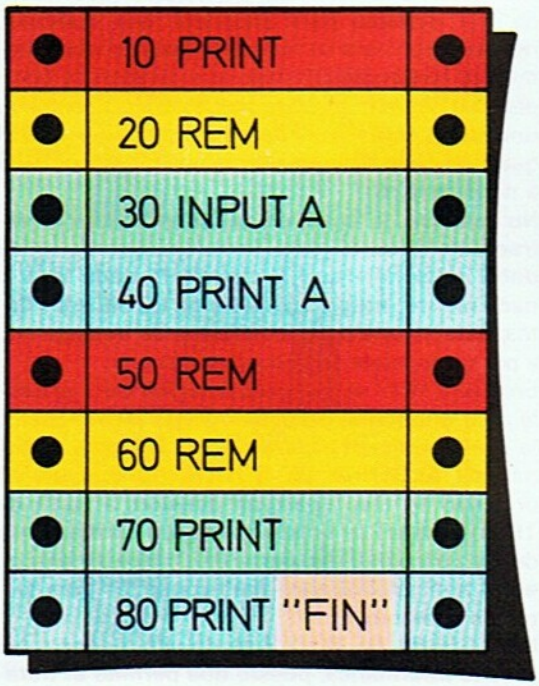
Para tener acceso a estos códigos de control es necesario situarse en el modo «extendido» (cursor **E** que se obtiene presionando simultáneamente las teclas **CAPS SHIFT** y **SYMBOL SHIFT**), y emplear los caracteres de la fila superior del teclado (los numéricos). Hecho esto, los dígitos del 0 al 7 establecen el color de papel correspondiente y, si se pulsan en combinación con **CAPS SHIFT**, el color de la tinta.

Continuando en el modo «extendido», podemos tener también acceso a las funciones de brillo e impresión intermitente. Pulsando **8** obtenemos el brillo normal, y con **9** el brillo especial. De forma similar, con **CAPS SHIFT 8** activamos la intermitencia y con **CAPS SHIFT 9** la desconectamos. Como podemos observar, los códigos a los cuales no se tiene acceso por este sistema, son los especiales: el de transparencia y el de contraste. Además de todo esto, puede también accederse al vídeo inverso, esta vez en el modo usual (cursores **K**, **C** o **L**), con **CAPS SHIFT + 4**, y desconectarlo a través de **CAPS SHIFT + 3**.

Todos estos caracteres de control son dobles, debido a lo cual, cuando es preciso eliminarlos con **DELETE (CAPS SHIFT+0)** hemos de pulsar la tecla dos veces. Por este mismo motivo, y dado que estos controles no son imprimibles, podemos notar ciertas anomalías en el desplazamiento del cursor sobre una línea con caracteres de control; concretamente, escuchar el «click» de pulsación de la tecla, pero no apreciar el avance del cursor. Como resumen final de todo lo dicho, será de utilidad un cuadro donde se describan las funciones de la fila superior del teclado, en los diferentes modos del cursor. (Cuadro 4)

Cualquier carácter de control introducido por el teclado es doble, y aunque no se encuentre representado, precisa de dos pulsaciones de DELETE (CAPS SHIFT + 0) *para su eliminación.*

Los caracteres de control de color obtenibles mediante el teclado, también se reflejan en los listados de los programas.



CONTROL POR EL TECLADO

MODO EXTENDIDO

SIN CAPS SHIFT	CON CAPS SHIFT
0 PAPER 0	0 INK 0
1 PAPER 1	1 INK 1
2 PAPER 2	2 INK 2
3 PAPER 3	3 INK 3
4 PAPER 4	4 INK 4
5 PAPER 5	5 INK 5
6 PAPER 6	6 INK 6
7 PAPER 7	8 FLASH 0
9 BRIGHT 1	9 FLASH 1

MODOS K, L Y C

CAPS SHIFT + 4 = INVERSE 1
CAPS SHIFT + 3 = INVERSE 0

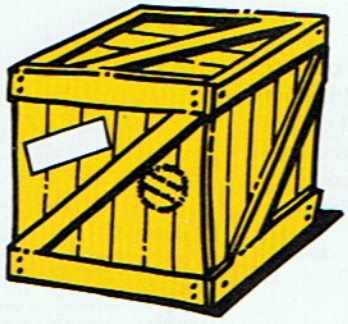
LOGICA INFORMATICA

TODO Sistema Ordenador está compuesto fundamentalmente por dos elementos: el *hardware* y el *software*.

El primero de ellos está conformado por el conjunto de elementos físicos que integran la máquina, tales como resistencias, transistores, memorias, microprocesador, etc... De hecho, el término inglés *hardware*, cuya traducción aproximada es «artículo de ferretería», designa de una manera un tanto despectiva a este potente conjunto de instrumentos electrónicos. Así pues, podemos decir que este elemento del Sistema constituye el SOPORTE FISICO del mismo.

No obstante, pese a la complejidad técnica de los integrantes del *hardware*, éste carece de utilidad alguna, y se convierte en una mera masa de hilos conductores, si no se le brinda un sistema de trabajo que le enseñe a realizar las tareas que deseamos, para las cuales está capacitado. Esto es algo que podríamos llamar SOPORTE LOGICO, y constituye el denominado *software*. Término que carece de un significado inglés concreto, y que en realidad nace de una forma un tanto chistosa: puesto que el *hardware* es el soporte «duro» *(hard* en inglés significa duro), su complemento ideal será el soporte «blando» o *software* (el significado literal de *soft* es blando).

A continuación analizaremos el funcionamiento interno, del *hardware*, pero para ello necesitamos unas nociones previas de la lógica empleada en la informática.

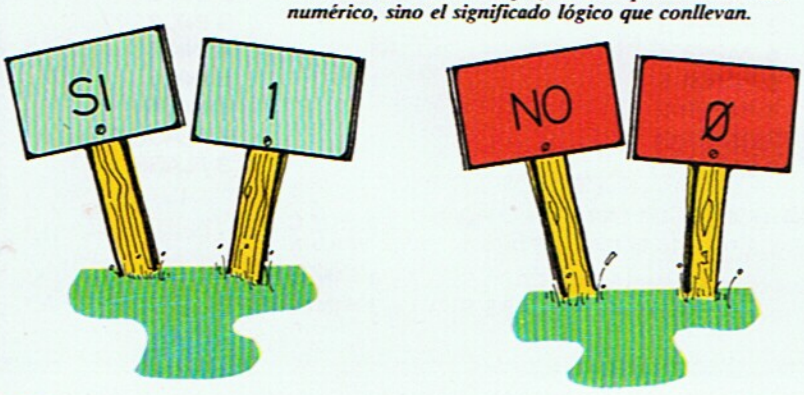


hardware

software

El soporte físico del ordenador se denomina hardware (artículo de ferretería); por oposición, el soporte lógico pasó a denominarse software.

Aunque los dos posibles valores lógicos se representen mediante los números 1 y 0, no es importante su valor numérico, sino el significado lógico que conllevan.



LOGICA MATEMATICA

Cuando construímos un sencillo sistema eléctrico, por ejemplo para encender una bombilla, nuestro problema se reduce a localizar unos elementos bastante comunes, que carecen de complejidad alguna: un trozo de cable, una bombilla, una pila y, sobre todo, un interruptor, que será el que nos permita encender o apagar la bombilla a nuestro gusto.

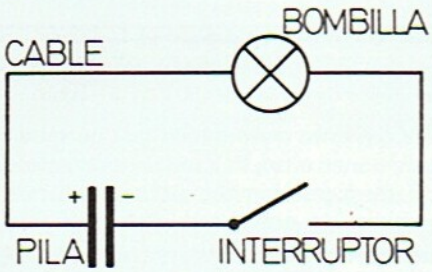
No obstante, la finalidad de un ordenador es extraordinariamente más compleja, y precisa de unos elementos que le permitan, nada más y nada menos, tratar de una forma LOGICA los datos; esto es una incipiente forma de inteligencia, y por tanto, nada sencillo de conseguir.

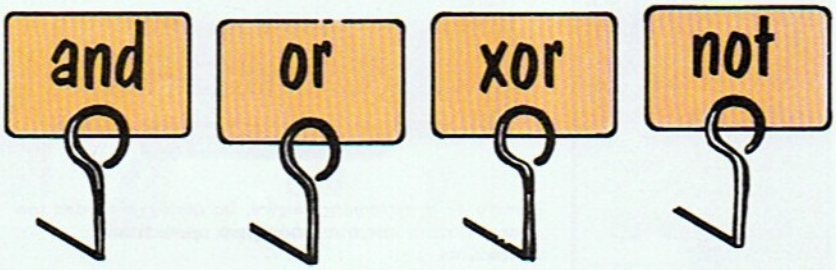
En el año 1847 el matemático inglés George Boole creó una forma particular de las matemáticas, la cual permite el tratamiento de la lógica, denominada ALGEBRA DE BOOLE. Aunque en un principio no tuvo una gran difusión, a partir de 1938 comenzó su utilización a gran escala, a raíz de su aplicación por parte de Claude Shannon, en el análisis de redes multicontactos para sistemas telefónicos.

El ALGEBRA DE BOOLE es de vital importancia para la informática, puesto que permite el trata-

miento matemático, y por tanto mediante circuitos electrónicos, de la lógica.
La lógica matemática se basa en el sistema binario, en el cual existen dos únicos dígitos: el cero y el uno. Para nosotros, cada uno de estos dígitos representa un ESTADO LOGICO diferente y opuesto.

En un esquema eléctrico simple como el del dibujo, el interruptor es el que nos permite determinar el estado de la bombilla.





Las operaciones lógicas que estudiaremos son AND, OR, XOR y NOT.

El «impulso vital» lo recibe el ordenador a través de la corriente eléctrica, y ésta se puede considerar también bajo un prisma binario: en determinado punto de un circuito hay corriente o no hay corriente, una bombilla está encendida o está apagada, etc...
Así pues, nada más fácil que estudiar los dos estados de la corriente eléctrica (hay o no hay) mediante el álgebra de Boole, destinada precisamente al tratamiento de dispositivos de este tipo: binarios o biestado.

INFORMACION LOGICA

Todos los dispositivos básicos del ordenador están preparados para tratar la información binaria: una información que únicamente puede ser un uno o un cero.
Ya hemos visto anteriormente, que si comenzamos a pensar en problemas concretos que poder tratar mediante el sistema binario, llegaremos a la conclusión de que algunos de ellos sí pueden ser representados por este método. Así por ejemplo, dos informaciones opuestas (uno y cero) son suficientes para contener una respuesta afirma-

BITS

Todo sistema ordenador está compuesto por dos elementos fundamentales: el *hardware* y el *software*.

*

En el álgebra de Boole no operamos con valores numéricos, sino con valores lógicos, los cuales pueden adoptar dos únicos estados.

*

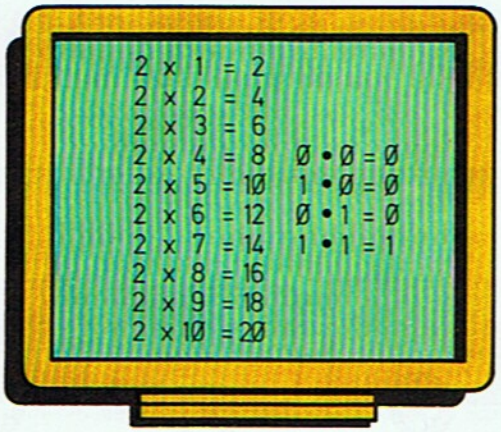
Los dos estados lógicos se representan mediante los dígitos 1 y 0.

BITS

En una operación lógica intervienen las variables lógicas (operandos), y la operación lógica (AND, OR, XOR o NOT), de forma que proporcionen un resultado lógico.

*

Al valor lógico uno **(1)** se le suele atribuir un significado positivo: sí, existencia, afirmación, cumplimiento de una condición, etc... El cero recibe siempre el valor opuesto.

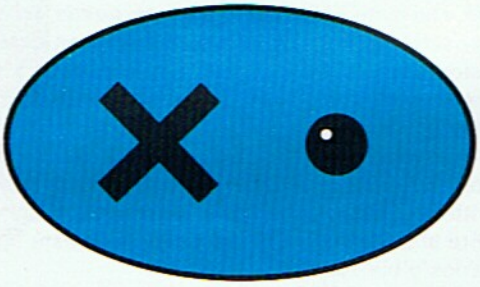


Dentro de la matemática lógica, las tablas de verdad son comparables a nuestras tablas para operaciones aritméticas.

tiva o negativa, si una bombilla está encendida o apagada, etc...

Ahora bien, la información que nosotros deseamos tratar es considerablemente más compleja, y no se reduce a «blanco o negro». Por tanto, agrupamos las unidades simples de información (bits), en conjuntos con más capacidad denominados bytes, y palabras (dos bytes). Tengamos mucho cuidado en no confundir el término informático palabra, con lo que los humanos entendemos como tal en nuestro lenguaje.

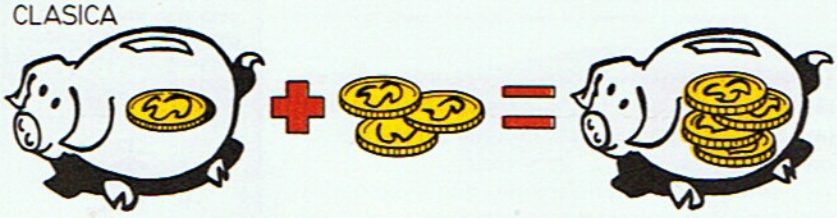
Bien, puesto que conocemos el sistema de estudiar informaciones de relativa complejidad, en base al tratamiento de simples unos y ceros, si conseguimos esto último, tendremos a nuestro alcance la posibilidad del tratamiento lógico de datos. A partir de este punto es donde entra el álgebra de Boole.



Debido a la similitud entre los resultados lógicos obtenidos por AND, y los que podrían obtenerse considerando sus operandos como valores numéricos, esta operación se conoce como PRODUCTO LOGICO.

Mientras que las operaciones aritméticas nos proporcionan resultados cuantificables, las operaciones lógicas nos reportan resultados lógicos.

CLASICA



LOGICA

ALGEBRA DE BOOLE

Este instrumento matemático nos permite realizar operaciones entre valores para obtener un resultado, de forma similar a como con nuestras matemáticas de andar por casa hacemos una suma para hallar el total de nuestros ahorros. Ahora bien, entre este tipo de operaciones a las cuales estamos acostumbrados y las que se contemplan en el álgebra de Boole, existen ciertas diferencias.

En primer lugar, tanto los datos a operar como los resultados tienen dos únicos valores posibles: cero o uno. En segundo lugar, las operaciones no están destinadas a obtener resultados de cantidad (total ahorrado), sino lógicos.

Asociada a esta segunda diferencia está la tercera: dado que los resultados a obtener son resultados lógicos, los valores que se operan (cero o uno) no tienen el sentido de cantidad numérica, sino de estado lógico.

Usualmente, para el estado lógico uno **(1)** adoptamos el significado del cumplimiento de una condición, la existencia de corriente, la bombilla encendida, la respuesta sí, etc... y el cero **(0)**, al no cumplimiento de la condición, la no existencia de corriente, la bombilla apagada, la respuesta no, etc...

Las operaciones del álgebra de Boole que estudiaremos se denominan AND (en inglés y), OR (o), XOR (o exclusivo) y NOT (no). En cuanto a los valores que se operan para obtener el resultado, toman el nombre de VARIABLES LOGICAS, y suelen estar representadas por las letras del abecedario (variable lógica A, variable lógica B, etc...), y por último, el resultado de la operación lógica recibe el nombre de RESULTADO LOGICO.

Del mismo modo que existen unas tablas de sumar, multiplicar, etc... el álgebra de Boole tam-

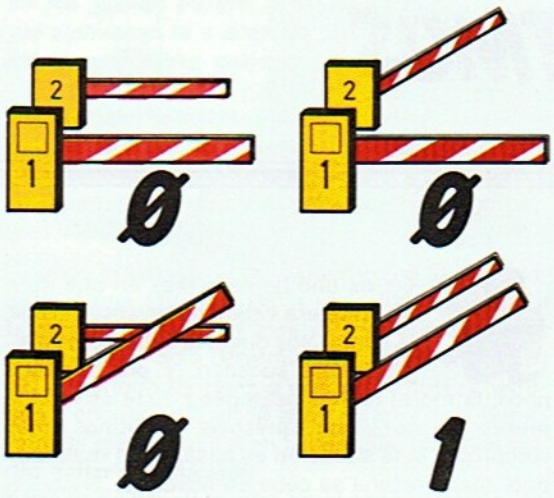
bién dispone de unas tablas de resultados de sus operaciones, denominadas TABLAS DE VERDAD. Así por ejemplo, si deseamos saber el resultado de la operación 3 por 2, consultamos la tabla de multiplicar y constatamos que es seis. Análogamente, si deseamos conocer el resultado de 1 AND 0, consultamos la tabla de verdad de la operación AND, y averiguamos que su resultado es cero.
Afortunadamente, las tablas de verdad son mucho más simples de aprender que las de operaciones convencionales, puesto que sólo utilizan ceros y unos, y no tiene más que cuatro combinaciones.
A continuación estudiaremos cada una de las operaciones separadamente.

AND

Se representa mediante un círculo relleno (●). En la siguiente tabla de verdad hemos representado con A y B dos variables lógicas cualquiera y mediante Q el resultado de la operación en cada caso.

A	●	B	Q
0	●	0	0
1	●	0	0
0	●	1	0
1	●	1	1

La operación AND también es conocida bajo el nombre de PRODUCTO LOGICO, debido a que el resultado de la operación coincide enteramente con los valores numéricos de una multiplicación clásica.
El hecho de que reciba el nombre de AND (en inglés Y) no es una mera coincidencia. Realmente, AND es el operador lógico que resuelve problemas del tipo SI <condición> Y <condición> ENTONCES ... <resultado>.
Veamos un ejemplo. Supogamos que deseamos cruzar un paso a nivel con dos barreras (una a cada lado de la vía); el que podamos pasar o no (resultado lógico) depende directamente de dos circunstancias (variables lógicas): que se encuentre levantada la barrera uno Y (AND) que se encuentre levantada la barrera dos.
Llamaremos A a la variable lógica "la barrera uno está levantada", que naturalmente puede adoptar dos valores: 1 (es cierto que la barrera uno está levantada) ó 0 (no es verdad que la barrera uno esté levantada). B será la otra variable lógica, correspondiente a "la barrera dos está levantada", que como en el caso anterior adoptará los valores 1 ó 0, según la variable sea cierta o no. El resultado de nuestro problema, denominado Q, será "podemos cruzar la vía", y como es natural, adoptará también dos valores: 1 si podemos cruzar, y 0 si no podemos. Ahora debemos encontrar la operación a utilizar. Puesto que es necesario que las variables A y B se cumplan para que se cumpla el resultado, nos encontramos ante una operación AND. Ya sólo nos queda dar valores a las variables y averiguar el resultado en cada caso.
Si la barrera uno no está levantada Y la barrera dos no está levantada; o lo que es lo mismo: A vale cero y B vale cero. Consultando en la tabla de verdad observaremos que 0 ● 0 = 0. Por lo tanto, no podemos pasar.
Si la barrera uno está levantada Y la barrera dos no está levantada; es decir, A vale uno Y B vale cero. Según la tabla de verdad 1 ● 0 = 0. Así pues, no podemos pasar. Sería terrible quedarnos entre las dos barreras y que entonces viniera el tren.
Si la barrera uno no está levantada Y la barrera dos está levantada; lo cual se puede expresar también de la siguiente forma: A vale cero Y B vale uno. Consultando en la tabla de verdad observaremos que 0 ● 1 = 0. Por lo tanto, no podemos pasar.
Si la barrera uno está levantada Y la barrera dos está levantada; lo cual significa que A vale uno Y B vale uno. Siguiendo la tabla de verdad diremos que 1 ● 1 = 1. Es decir, podemos pasar puesto que ninguna de las barreras nos lo impide.



Para que podamos atravesar un paso a nivel con dos barreras, se tiene que cumplir una estructura lógica del tipo AND.

¡!

El *hardware*, término inglés cuyo significado es «artículo de ferretería», constituye el soporte físico del ordenador: su circuitería, memorias, resistencias y todos sus demás componentes.

*

El término inglés *software*, carente de un significado literal, y que se acuña por oposición a *hardware* (*hard* significa «duro» y *soft*, «blando»), designa al soporte lógico del ordenador: el conjunto de la programación que permite convertir el *hardware* en un sistema ordenador.

*

La lógica empleada en la informática, se rige por las reglas matemáticas enunciadas por George Boole, conocidas como ALGEBRA DE BOOLE.

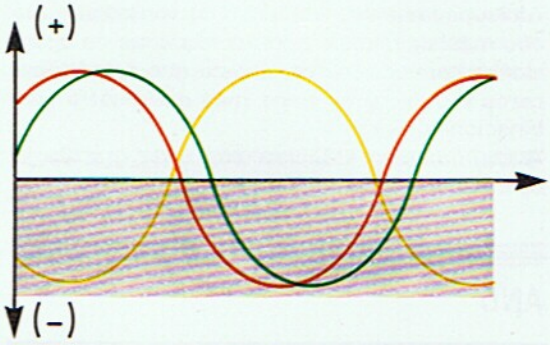
BIORRITMOS

HOY es uno de esos días en que más nos hubiera valido no levantarnos de la cama. Nos duele la cabeza, los disgustos se suceden, y por si fuera poco, la moral no nos acompaña. Este es el momento de consultar nuestros biorritmos para comprobar si la situación es pasajera, o realmente la mala fortuna se ceba en nosotros.

Desde el mismo momento de nuestro nacimiento, cuando el cuerpo humano comienza a vivir con independencia, se inician una serie de ciclos vitales que nos acompañarán durante toda nuestra existencia. Todos estos ciclos se repiten en determinados períodos de tiempo fijos, siguiendo siempre unos mismos índices de ascenso y caída, de forma que una vez conocido su punto inicial (el día de nacimiento), y el tiempo transcurrido desde ese momento, podemos calcular el punto exacto en que se encuentra cualquiera de los ciclos.

Los biorritmos se basan fundamentalmente en tres ciclos vitales: ciclo físico, ciclo emocional y ciclo intelectual. Todos ellos, como hemos dicho anteriormente, tienen su origen común en el día de nuestro nacimiento, y su duración exacta es la siguiente:

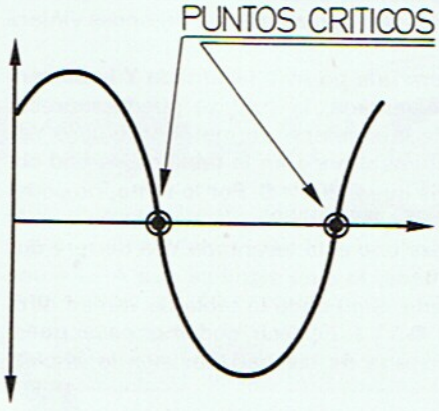
Ciclo físico - 23 días.
Ciclo emocional - 28 días.
Ciclo intelectual - 33 días.

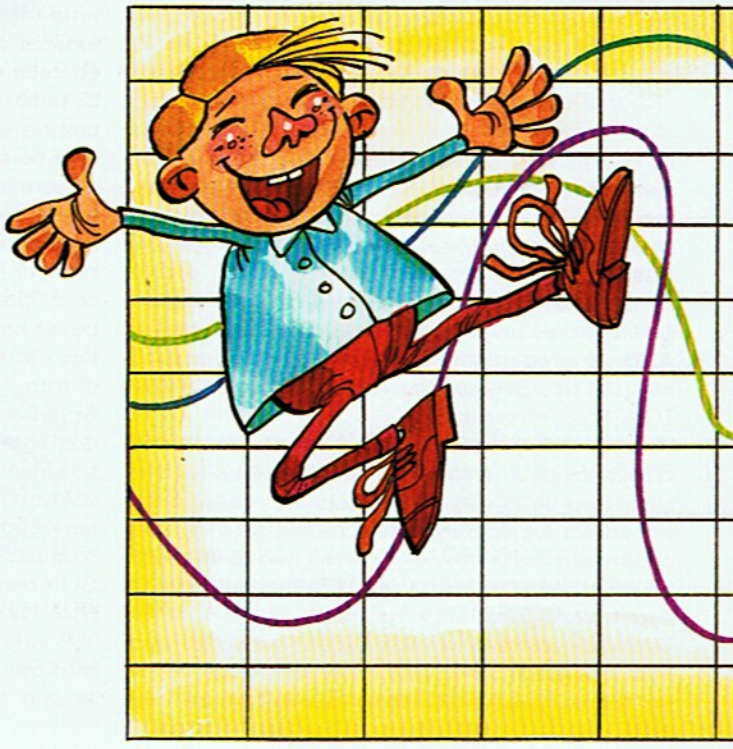


En la zona negativa del biorritmo, nos encontramos en un mal momento para acometer tareas en el área representada.

Esto quiere decir, que tomando como punto de partida (nacimiento) el cero, y estudiando, por ejemplo, el ciclo físico, observaremos que se produce una pendiente de ascenso cuya duración es siete días, la cual va seguida de un descenso de catorce días, a cuya mitad volveremos a atravesar el punto cero, pero en esta ocasión descendiendo; por último, el ciclo se cerrará volviendo

Los puntos críticos son los más importantes dentro de la interpretación de los biorritmos, y corresponden a la intersección de las curvas con el eje central.





BITS

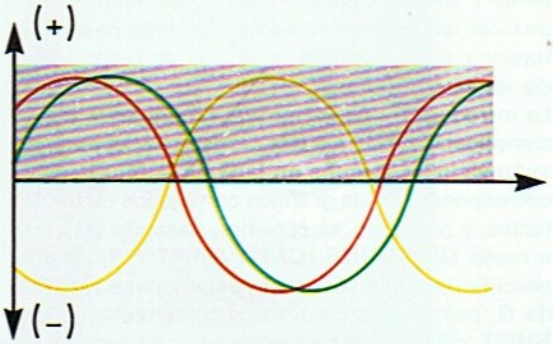
La introducción de fechas ha de ser como cada uno de los ejemplos que aparecen en la pantalla.

*

El año máximo que admite el programa es el 9999.

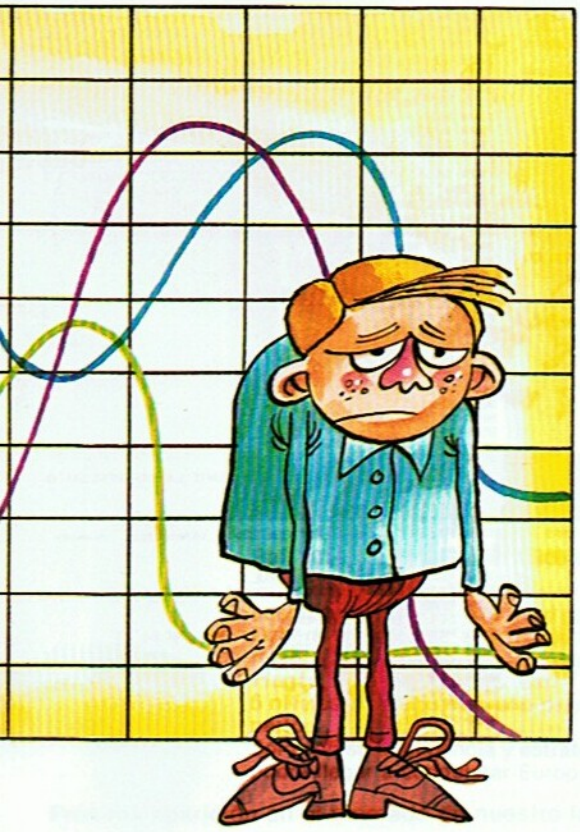
*

A los puntos de intersección de las curvas con el eje central se les denomina PUNTOS CRITICOS.



En la zona positiva de las curvas, nuestra predisposición hacia las tareas que representan es buena.

a un ascenso, que en los siete días restantes nos situarán de nuevo en la posición de inicio.
Técnicamente, podemos decir que la representación del ritmo biológico está determinada por una sinusoide, de amplitud comprendida entre los valores +1 y –1, correspondientes a la fórmula: **Amplitud = SENO (360 × ND / DC)**; donde **ND** es el número de días después del nacimiento, y **DC** es la duración de cada ciclo.



Cuando la sinusoide se encuentra en la zona positiva (zona superior de la pantalla), indica una buena predisposición ante el área que señala la curva de un ciclo determinado. Si por el contrario el período de tiempo que nos interesa estuviese encuadrado en la zona inferior que determina el eje horizontal en la pantalla, más nos valdrá andar con cuidado.
Esto no quiere decir que en el caso de una mala racha de nuestro ciclo físico, vayamos a enfermar, pero sí es más fácil que nos sintamos cansados o seamos más propensos a alguna infección.
Del mismo modo, atravesar una buena época intelectual, nos afecta en el sentido de reportarnos menor trabajo estudiar un mismo tema, o comprender determinado problema; un estado emocionalmente bajo, nos hace propensos a las depresiones, o convertir un problema que podía ser un granito de arena en una auténtica montaña.
Los entendidos en el tema, se decantan fundamentalmente hacia la opinión de que más importantes que las zonas positivas o negativas de los ciclos, son los denominados puntos críticos: aquellos en los cuales la curva atraviesa el límite entre las dos zonas, es decir, se encuentra en el que hemos denominado punto cero.
Durante estos puntos críticos somos especialmente sensibles, dado que nos encontramos en una fase de transición entre unos estados y otros, y por tanto, en una situación de claro desequilibrio. Así pues, será tanto más delicado el momento, cuantos más puntos críticos coincidan. Es relativamente frecuente que dos ciclos atraviesen un punto crítico el mismo día, en cuyo caso nuestro estado de desequilibrio puede jugarnos una mala pasada.
No obstante, la coincidencia de tres puntos críticos, que lógicamente se dio el día de nuestro nacimiento, cosa por otra parte bastante clara, puesto que en el preciso momento de nacer atravesamos el punto más crítico de nuestra vida, no se repite hasta cincuenta y siete años más tarde.
La teoría de los biorritmos es seguida desde hace varios años por distintas entidades de países tan desarrollados como Japón, Estados Unidos, Rusia e inclusive España, para mejorar la eficacia en el trabajo.
En este sentido, es ya conocido el caso de una compañía de transportes japonesa, la cual averiguó mediante un estudio estadístico, que los accidentes sufridos por sus conductores solían

BITS

Los caracteres que aparecen en el listado subrayados doblemente, deben introducirse como los gráficos cambiados (con **CAPS SHIFT**) de las teclas correspondientes.

*

Los literales subrayados **INV.** y **TRUE**, encerrados entre corchetes, se corresponden a los códigos de control **INVERSE VIDEO** y **TRUE VIDEO**, obtenidos por el teclado mediante **CAPS SHIFT + 4** y **CAPS SHIFT + 3**, respectivamente.

*

Para la grabación del programa teclearemos el siguiente comando: **SAVE "BIO"**. En el caso de que lo deseáramos almacenar con autoejecución, utilizaremos la siguiente instrucción: **SAVE "BIO" LINE 10.**

coincidir con sus puntos críticos físicos, en los cuales la lentitud de reflejos, y el cansancio superior al normal los hacían propensos a los percances.

A raíz de este descubrimiento, la compañía decidió advertir a sus empleados de los días en que se encontraran bajo estas circunstancias: ¡era más barato poner sobre aviso, que tener que abonar costosas reparaciones! Ello redujo en una proporción escalofriante el número de accidentes.

Una vez cargado y ejecutado correctamente el programa, aparecerán en la pantalla unos ejes de coordenadas y dos secuencias de introducción de datos. En la primera de ellas, teclearemos nuestra fecha de nacimiento, y en la siguiente, el día, mes y año a partir de la cual queremos nuestro biorritmo.

La manera en que debemos introducir las fechas es muy sencilla y está explicada en el propio programa a modo de ejemplo. En primer lugar deberemos indicar el día de nacimiento, completando con un cero por la izquierda si éste tuviera un solo dígito. A continuación, y sin ninguna separación intermedia, hemos de introducir el mes, para lo cual utilizaremos las tres iniciales correspondientes (en mayúsculas). Por último, añadiremos a la cadena de introducción las cuatro cifras del año.

El último dato que requiere el programa es la fecha actual, para así poder calcular el número de días transcurridos desde nuestro nacimiento. Tan sólo nos queda esperar a que nuestro Spectrum dibuje cada una de las sinusoides correspondientes a cada ciclo, dentro de un período de 30 días. Para ello, existe un período previo de cálculo, pasado el cual se accede a la fase de trazado de curvas. Estas se representarán en los colores rojo, verde y amarillo, para el ciclo físico, emocional e intelectual, respectivamente. En todo caso, una leyenda en la pantalla, indica la correspondiencia de estos colores.

La introducción del programa no presenta mayor complicación, que los dos caracteres doblemente subrayados presentes en la línea 8605 (3); estos corresponden a los gráficos cambiados de dichas teclas, y por tanto, se obtienen pasando el cursor a modo **GRAPHICS (CAPS SHIFT + 9)**, lo cual percibiremos por el cambio del mismo a la forma de **G**, pulsando a continuación las teclas **CAPS SHIFT** y **3** de forma simultánea.

Por otra parte, los literales subrayados **INV.** y **TRUE** encerrados entre corchetes, corresponden con los caracteres de control **INVERSE VIDEO** y **TRUE VIDEO**, respectivamente. Al encontrarse en este caso dentro de líneas **REM**, no es estríctamente necesaria su presencia, y tienen la única finalidad de resaltar en la pantalla dichas líneas, frente a las del resto del listado.

```
  10 REM ***********************
  20 REM * J.M.MAYORAL SERRANO *
  30 REM ***********************
  40 REM *  Biorritmos   1985  *
  50 REM ***********************
  60 BORDER 0: PAPER 0: INK 9
  65 LET SW1=0
  70 CLS
  80 DIM A(150)
  90 LET COL=1
 150 GO TO 700
 200 REM [INV.] DIBUJO EJES [TRUE]
 210 PLOT 10,165
 220 DRAW 0,-101
 230 PLOT 10,114
 240 DRAW 240,0
 250 FOR N=18 TO 256 STEP 8
 260 PLOT N,115
 270 DRAW 0,-2
 280 NEXT N
 290 PRINT AT 0,0;"(+)"
 300 PRINT AT 14,0;"(-)"
 305 REM [INV.]  FONDO  [TRUE]
 310 FOR N=0 TO 14
 320 PRINT  PAPER 1: OVER 1;AT N,0;"
                                    "
 330 NEXT N
 340 RETURN
 500 REM [INV.] ITRODUCCION DATOS [TRUE]
 510 PRINT AT 16,0; PAPER 6;" INTRODUCE FECHA DE NACI
MIENTO ."
 520 PRINT TAB 8;"( DD/MMM/AAAA )"
 525 PRINT
 530 PRINT TAB 6;"Ejemplo:  07ENE1964"
 540 INPUT "Fecha nacim.",F$
 542 IF LEN F$<>9 THEN  LET SW=0: RETURN
 545 IF CODE F$(1)<48 OR CODE F$(1)>57 THEN  LET SW=0
: RETURN
 550 IF CODE F$(2)<48 OR CODE F$(2)>57 THEN  LET SW=0
: RETURN
 555 IF VAL F$( TO 2)>31 OR VAL F$( TO 2)<=0 THEN  LE
T SW=0: RETURN
 560 RESTORE 7010
 565 FOR N=1 TO 12
 570 READ A$,A
 575 IF A$=F$(3 TO 5) AND A>=VAL F$( TO 2) THEN  GO T
O 600
 580 NEXT N
 590 LET SW=0: RETURN
 600 IF CODE F$(6)<49 OR CODE F$(6)>57 THEN  LET SW=0
: RETURN
 605 IF CODE F$(7)<48 OR CODE F$(7)>57 THEN  LET SW=0
: RETURN
 610 IF CODE F$(8)<48 OR CODE F$(8)>57 THEN  LET SW=0
: RETURN
 615 IF CODE F$(9)<48 OR CODE F$(9)>57 THEN  LET SW=0
: RETURN
 620 LET SW=1
 625 RETURN
 700 REM [INV.]  PROGRAMA PRINCIPAL [TRUE]
 710 GO SUB 200
 720 GO SUB 500
 730 IF NOT SW THEN  GO TO 720
1010 LET DN=VAL F$( TO 2)
1020 LET AN=VAL F$(6 TO )
1030 LET N$=F$(3 TO 5)
1035 LET B$=F$
1050 GO SUB 8000
1060 PRINT AT 16,0; PAPER 2;" INTRODUCE FECHA PARA CO
MIENZO          DE TU BIORRITMO
1070 PRINT TAB 8;"( DD/MMM/AAAA )"
1080 PRINT
1090 PRINT TAB 6;"Ejemplo:  09MAY1985"
1100 INPUT "Fech/inic. BIORRIT. ";F$
1110 GO SUB 542
1120 IF NOT SW THEN  GO TO 1110
1130 LET DI=VAL F$( TO 2)
1140 LET AA=VAL F$(6 TO 9)
1150 LET M$=F$(3 TO 5)
1155 LET C$=F$
1160 GO SUB 8000
1170 GO SUB 7000
1180 PRINT AT 16,0;"FECHA NACIMIENTO: ";DN;"/";N$;"/"
;AN
1185 PRINT
1190 PRINT "FECHA / COMIENZO"
1200 PRINT "DEL BIORRITMO   : ";DI;"/";M$;"/";AA
1210 PRINT
1220 PRINT "DURACION BIORRIT.:    30 DIAS"
1225 FOR F=1 TO 30: PAUSE 10: NEXT F
1226 BEEP .1,40
1227 GO SUB 8000
1230 REM [INV.] CALCULOS [TRUE]
1231 PRINT AT 16,0
1232 RESTORE 8595
1235 FOR D=23 TO 33 STEP 5
1237 PRINT AT 21,0; FLASH 1; INK 2; PAPER 7;"       C
A L C U L A N D O
1238 LET C=1
1240 FOR N=0 TO 29 STEP .2
1250 LET A(C)=SIN ((2*PI*(ND+N))/D)
1260 LET C=C+1
1280 NEXT N
1290 PRINT AT 21,0; FLASH 1; INK 9; PAPER 4;"       D
 I B U J A N D O
1300 GO SUB 8600
1310 NEXT D
1320 PRINT AT 21,0;"                                  "
1330 IF INKEY$="" THEN  GO TO 1330
1340 RUN
7000 REM [INV.]SBR. CALCULO DIAS VIV.[TRUE]
7010 DATA "ENE",31,"FEB",28,"MAR",31,"ABR",30,"MAY",3
1
7020 DATA "JUN",30,"JUL",31,"AGO",31,"SEP",30,"OCT",3
1,"NOV",30,"DIC",31
7030 RESTORE 7010
7040 LET DA=AA-AN
7045 IF NOT DA THEN  GO TO 8500
7050 LET ND=DA*365
7060 FOR K=1 TO 12
7070 READ A$,A
7080 IF A$=N$ THEN  LET ND=ND-DN: GO TO 7110
7090 LET ND=ND-A
7100 NEXT K
7110 LET DV=ND
7120 RETURN
7990 REM [INV.]  SBR.BORRADO  [TRUE]
8000 FOR Q=15 TO 21
8010 PRINT AT Q,0;"
                                    "
8020 NEXT Q
8030 RETURN
8500 REM
8510 REM
8520 CLS
8530 PRINT AT 10,2;"NO ES SUFICIENTE LA DIFERENCIA  D
E ANOS"
8540 PRINT
8550 PRINT
8560 PRINT "  Pulse RUN <ENTER> e introduzca  nuevas
fechas."
8570 STOP
8590 REM [INV.] GRAFICA [TRUE]
8595 DATA "FISICO",2,"EMOCIONAL",4,"INTELECT.",6
8600 READ A$,A
8601 LET C=1: LET COL=COL+1
8605 PRINT AT 14+COL,0;"CICLO ";A$, INK A;"33"
8610 FOR X=10 TO 250 STEP 2
8620 LET Y=114+(50*A(C))
8630 LET C=C+1
8635 CIRCLE  INK A; X,Y,1
8655 NEXT X
8660 RETURN
```

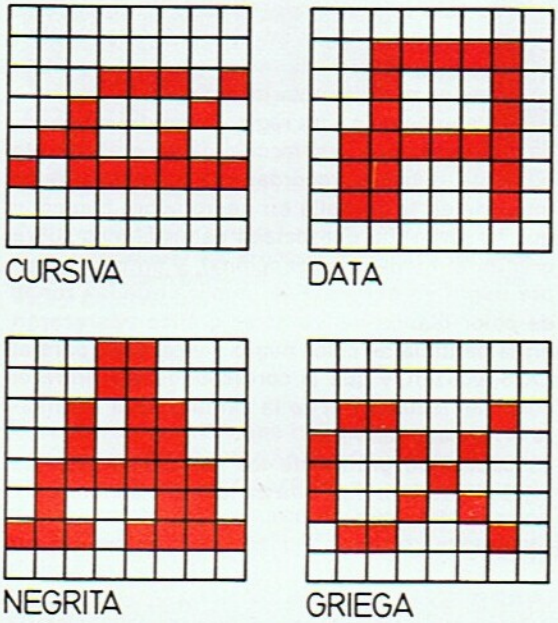
CARACTERES GRAFICOS

NO cabe duda de que el Spectrum cuenta con interesantes posibilidades en el campo de la representación de gráficos en la pantalla. Ya conocemos por comentarios anteriores, la existencia de los gráficos predefinidos por el Sistema, que son los contenidos en los números de la primera fila del teclado (1 a 8).

Pero como veremos más adelante, el ordenador nos da la facilidad de definir también nuestros propios caracteres. Esta es realmente una cualidad muy a tener en cuenta, puesto que nos permite la presentación altamente estética de todo tipo de programas: desde los técnicos, que precisan de caracteres especiales para su simbología (letras griegas, subíndice y superíndices, etc...), hasta los de juego, en los cuales podremos definir nuestros propios personajes, naves espaciales, etc... además del acceso a la escritura en otros idiomas, pudiendo redefinir ciertos caracteres especiales, como las vocales acentuadas francesas, las diéresis germánicas, etc...

Por último, una técnica muy especial, conocida como REUBICACION DEL AREA GENERADORA DE CARACTERES, nos permitirá la redefinición de alfabetos completos (cirílico, griego, etc...) o bien la remodelación del habitual, obteniéndose distintos tipos de letra: cursiva, data o magnetic, princetown, etc... o cualquier otra a nuestro antojo.

En primer lugar, estudiaremos en profundidad la utilización de los gráficos predefinidos, aquellos que ya conocemos, para a continuación pasar a ver los sistemas de generación de nuestros propios caracteres.



La reubicación del generador de caracteres nos permitirá la generación de alfabetos completos.

Los caracteres definibles por el usuario (U.D.G.) nos dejan una puerta abierta para la creación de simbología especial (subíndices, superíndices, acentos, diéresis, etc...).

GRAFICOS PREDEFINIDOS

Este juego de ocho caracteres gráficos, se ve en la realidad desdoblado en dieciséis, puesto que podemos obtener, con la sola pulsación de una tecla, los inversos de ellos *(INVERSE VIDEO)*. Para acceder a este juego de gráficos convencional, debemos recordar situarnos previamente en

¡!

Los gráficos predefinidos se desdoblan en 16 caracteres: 8 en vídeo normal, y 8 en vídeo inverso.

*

Para acceder al juego de caracteres gráficos, hemos de situar el cursor en el modo **GRAPHICS**, que se obtiene pulsando, o bien la tecla **GRAPH** (en el Plus), o bien las teclas **CAPS SHIFT + 9** (en el ZX Spectrum).

*

Los gráficos predefinidos son aquellos caracteres con códigos comprendidos entre el 128 y el 143.

el modo gráfico (cursor en modo **G**), mediante la pulsación simultánea de **CAPS SHIFT + 9**, para después introducir, con o sin **CAPS SHIFT**, cualquiera de los caracteres numéricos del 1 al 8. Dentro del modo **G**, los gráficos con **CAPS SHIFT** adoptan la configuración que aparece serigrafiada en blanco sobre las teclas; sin **CAPS SHIFT**, se accede al segundo juego, compuesto por los inversos del primero. Esto que acabamos de afirmar es sólo válido para el modelo ZX Spectrum, si bien no para el Plus, en el cual las correspondencias de pulsación de teclas y carácter gráfico obtenido es exactamente la contraria.

Si nos atenemos a esta regla, no tendremos nunca problemas con la selección de los gráficos. No obstante, debemos recordar que normalmente se imprime en la pantalla en negro sobre blanco, y que la serigrafía del teclado es en blanco sobre gris en el modelo convencional, y en blanco sobre negro en el Plus. Esto implica que las zonas de color blanco del carácter gráfico aparecerán en la pantalla de color negro y viceversa, para el ZX Spectrum, y que la correspondencia entre el color del teclado y el de la pantalla será la directa en el caso del Plus.

La capacidad gráfica de los caracteres gráficos predefinidos, va más allá de lo que a simple vista podemos apreciar; tan solo un poco de experiencia en su manejo, nos puede reportar sorpresas muy agradables.

Su utilización más común se da en la presentación de mensajes de grandes dimensiones en la pantalla. Por otra parte, estos caracteres se pueden emplear también para la representación de funciones en baja resolución, es decir, aquellas en las cuales cada punto representado coincide con un cuarto de carácter.

Como es lógico, los gráficos predefinidos forman parte integrante del juego de caracteres, debido a lo cual tienen asignado un código determinado. Gracias a ello, estos caracteres pueden ser obtenidos no sólo por el teclado, sino también a través de un programa, mediante la función **CHR$**. Los códigos de los caracteres gráficos predefinidos van desde 128 al 143. El siguiente miniprograma nos los muestra en la pantalla, acompañados de su código correspondiente.

```
10 PRINT "CODIGO","CARACTER"
20 PRINT "------","--------"
30 FOR I=128 TO 143
40 PRINT "  ";I,"     ";CHR$ I
50 NEXT I
```

En el programa podemos observar claramente la ventaja que supone poder emplear la función **CHR$**, puesto que ello nos permite utilizar una sola línea de programa para la impresión de caracteres (la 40), utilizando una variable (I) para designar el carácter concreto a representar.

El modo de cursor G y las teclas numéricas del 1 al 8, dan acceso a los gráficos predefinidos.

PECULIARIDADES DE LOS CARACTERES GRAFICOS

Los caracteres gráficos no pertenecen al código A.S.C.I.I. estándar, y por tanto, la mayoría de las impresoras, a excepción de las especialmente di-

El juego de ocho caracteres predefinidos se desdobla en dieciseis.

TECLA / CAMBIO	1	2	3	4	5	6	7	8
CON CAPS SHIFT	▟	▜	▬	▙	▌	▚	▗	■
SIN CAPS SHIFT	▙	▗	▬	▟	▐	▞	▜	□

señadas para nuestro ordenador, carece de la posibilidad de impresión de estos caracteres. Debido a ello, a la hora de realizar los listados en papel de los programas, se utiliza un convenio que nos interesa conocer; ya sea para poder interpretar los listados hecho por otros, fundamentalmente los aparecidos en publicaciones, como para utilizarlos en nuestros propios programas que deseemos enviar a concursos, o a nuestros amigos.

Dado que en un listado nunca puede aparecer, por la vía normal, un carácter subrayado, este tipo de distinción es la que se emplea para designar los gráficos. Así pues, siempre que veamos en un listado un carácter subrayado, debemos interpretar que se trata del gráfico de la tecla cuyo carácter principal aparece subrayado.

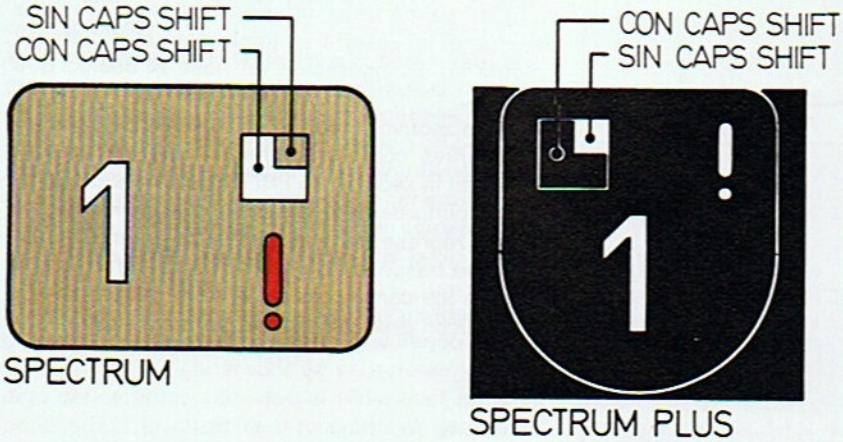
Así por ejemplo, un número cinco (5) subrayado, equivale al gráfico que se obtiene por la pulsación de la tecla **5.** Por tanto, para su introducción deberemos proceder de la siguiente forma:

— Pasar el cursor a modo gráfico (cursor **G**). En el Spectrum Plus existe una tecla especial a tal fin **(GRAPH)**, mientras que en el caso del ZX Spectrum es necesaria la pulsación de dos teclas simultáneamente: **CAPS SHIFT + 9.**

— A continuación, pulsaremos la tecla cuyo carácter principal aparece subrayado. En el caso concreto de nuestro ejemplo, deberemos pulsar la tecla **5**, lo cual hará aparecer en la pantalla no dicho dígito, sino su carácter gráfico correspondiente.

— Por último, y en caso de que no haya más caracteres gráficos consecutivos que introducir, deberemos restablecer el modo normal del cursor, ya sea L o C, para poder continuar con la introducción del listado. Para ello, deberemos pulsar de nuevo las teclas **CAPS SHIFT + 9**, o bien únicamente la del **9.** En el caso del modelo Plus, a esta posibilidades se añade la de pulsación de la tecla **GRAPH.**

Hasta ahora el sistema de subrayado ha cumplido plenamente su cometido, pero los problemas llegan de nuevo cuando deseamos representar los caracteres gráficos predefinidos en vídeo inverso, es decir, aquellos que se obtienen en el modo gráfico mediante la pulsación simultánea de una tecla numérica (de 1 a 8) y la tecla **CAPS SHIFT.**

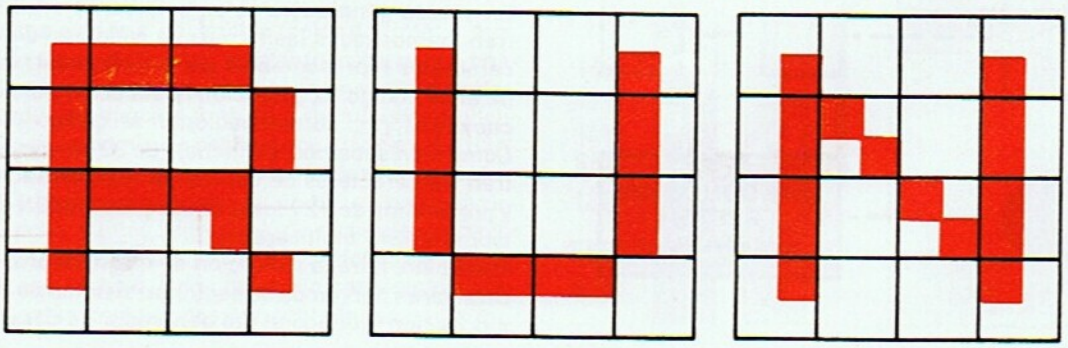


La correspondencia entre la serigrafía del teclado y el gráfico predefinido que se obtiene mediante la pulsación de las teclas correspondientes, es inversa en los modelos Plus y ZX Spectrum.

Este tipo de gráficos que habitualmente se conocen como GRAFICOS CAMBIADOS *(SHIFT* en inglés se traduce por CAMBIO), se representan de forma similar a los anteriores, pero esta vez con un subrayado doble. Así por ejemplo, un carácter 8 doblemente subrayado en un listado, debe ser introducido pasando el cursor a modo **GRAPHICS** (cursor G) y pulsando simultáneamente las teclas **CAPS SHIFT + 8.** Nunca olvidemos desconectar el modo gráfico al finalizar la introducción de una serie de los mismos, es decir, cuando el próximo carácter a introducir ya no deba ser considerado como gráfico.

A continuación adquiriremos los conocimientos sobre la forma en que el Spectrum almacena el diseño de los caracteres en su memoria. La zona de memoria destinada a tal fin, se denomina AREA GENERADORA DE CARACTERES. Pues bien, es interesante destacar que los caracteres gráficos predefinidos no se encuentran en dicha

Los gráficos predefinidos se suelen utilizar para la presentación de mensajes en grandes dimensiones.

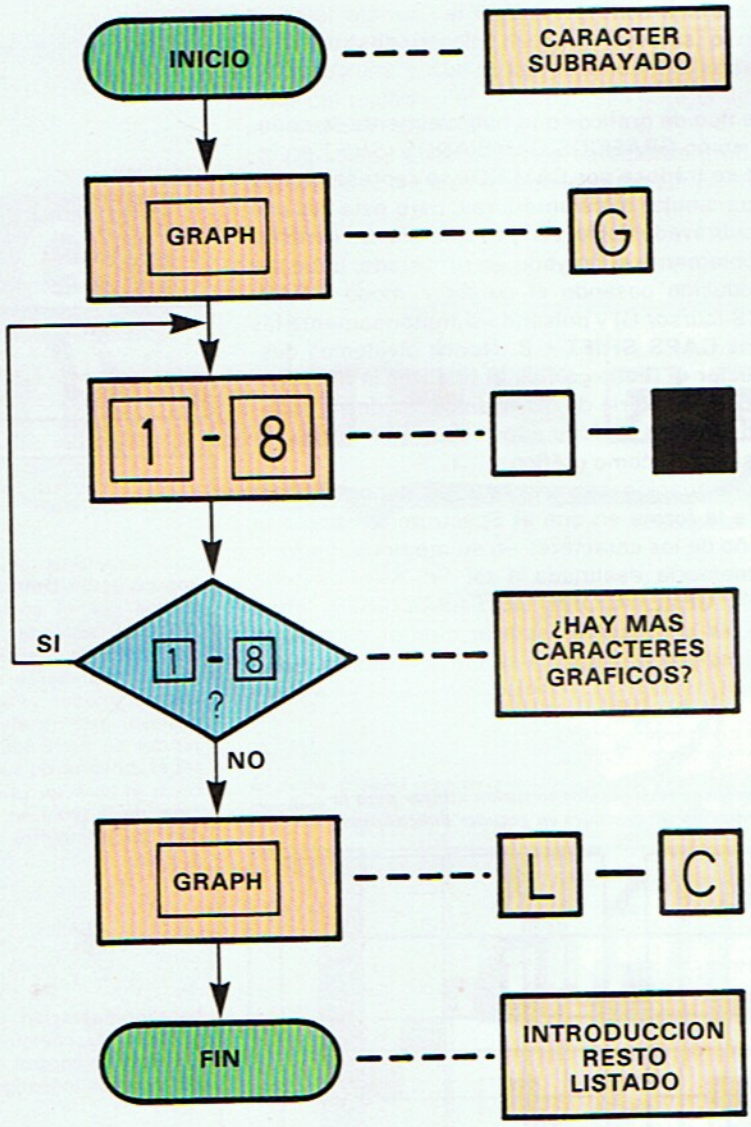


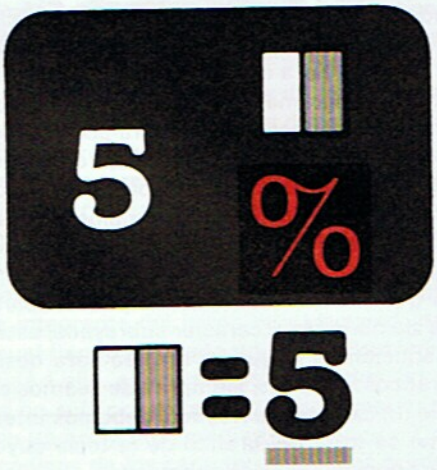
Los caracteres gráficos no están contemplados por el código A.S.C.I.I. estándar, y por tanto, la mayoría de las impresoras no los reconocen. Para soslayar este inconveniente, se suele adoptar el convenio de subrayar el carácter principal de la tecla en la cual se encuentre el gráfico.

La representación en pantalla de cualquier carácter de control es un signo de interrogación.

zona, ni realmente en ninguna otra, sino que cada vez que es necesaria su impresión, el ordenador calcula su configuración en base al código que reciben.
Por este motivo, algunas subrutinas de caracteres gigantes, como la que hemos utilizado en ocasiones en la sección de PROGRAMA (Psion Computers, cinta de demostración HORIZONTES), no pueden representar este tipo de caracteres, puesto que se basan en la ampliación de la forma que adoptan los caracteres en el área generadora, y los gráficos predefinidos no se hayan reflejados en ella.

Secuencia de introducción de un carácter subrayado.

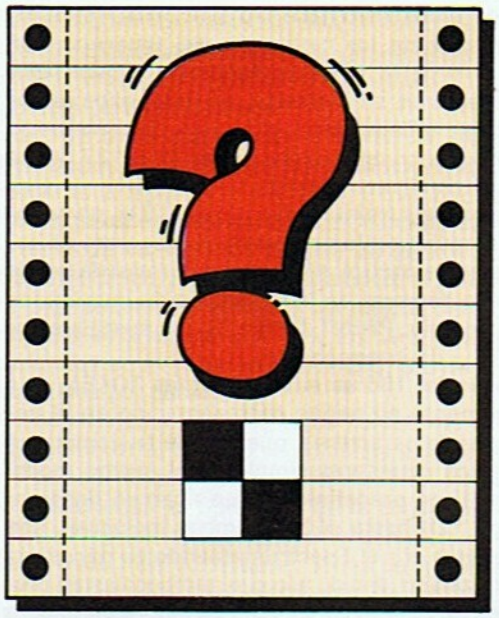




Cuando en un listado encontremos un carácter subrayado, habremos de introducir el gráfico de la tecla cuyo carácter principal se subraya.

EL GENERADOR DE CARACTERES

Al hablar de la composición de la pantalla, dijimos que es similar a la de un papel milimetrado, en el cual podemos contar 24 filas de 32 cuadrículas. Digamos también, para los iniciados en el álgebra, que es como una matriz de 24 filas por 32 columnas.
En cualquier caso, cada una de las 768 (24×32) posiciones de carácter en las cuales se divide la pantalla, consta a su vez de una configuración de puntos; concretamente se trata de una cuadrícula de 8 por 8, es decir, 64 puntos en total. Esta cuadrícula elemental de 8 por 8, permite la representación del juego completo de caracteres del Spectrum: alfabéticos, numéricos y gráficos, donde el atributo **INK** afecta a los puntos encendidos del carácter, y el atributo **PAPER** a los apagados.
En el área generadora de caracteres, se encuentran memorizadas las formas de cada uno de los caracteres representables por el ordenador: desde el de código 32 (espacio), hasta el símbolo del *copyright* (©), correspondiente al código 127. Como bien sabemos, por debajo de 32 se encuentran los caracteres de control no representables, y por encima de 127 los gráficos y los TOKEN (palabras BASIC multicarácter).
Pues bien, para la definición de nuestros propios caracteres habremos de seguir un sistema de memorización idéntico al del generador de caracte-



La mayoría de las impresoras, no diseñadas específicamente para nuestro aparato, ignoran los caracteres gráficos.

res, debido a lo cual hemos de estudiar previamente éste.

En primer lugar, diremos que el generador se encuentra en los últimos 3/4 K de la memoria ROM (768 bytes), la reservada al Sistema, en la cual nunca podemos escribir y que nunca se borra, ni tan siquiera apagando el ordenador. Esto es algo muy sencillo de intuir, dado que si no fuera así, una vez apagado el Spectrum éste no sabría, al encenderlo nuevamente, la forma de las letras, y no podría escribir caracteres en la pantalla para comunicarse con nosotros.

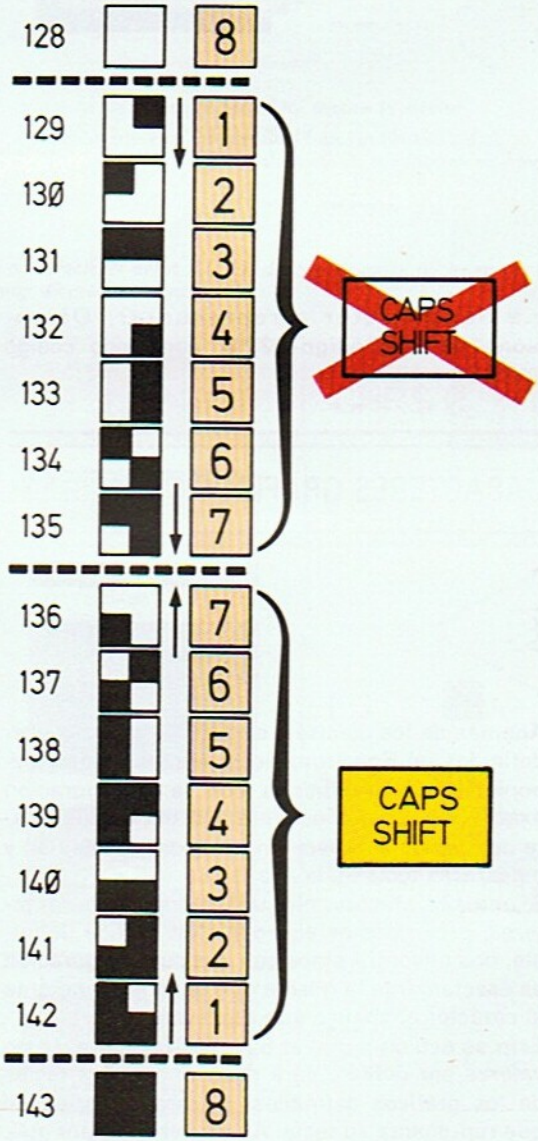
Así pues, el generador de caracteres se encuentra entre las direcciones de memoria 15616 (3D00 en base hexadecimal) y 16383 (3FFF hexadecimal), ambas inclusive. Este dato será muy importante cuando estudiemos el sistema de reubicación del generador de caracteres.

La forma en la cual se encuentran almacenados los caracteres es bastante fácil de asimilar. Cada carácter está compuesto por ocho líneas de ocho puntos, cada uno de los cuales puede adoptar dos únicos estados: encendido (color de tinta) o apagado (color de fondo). Por tanto, cada punto puede ser representado mediante una unidad mínima de información (bit), para lo cual se ha adoptado el convenio de designar con un cero los puntos apagados, y con un uno los encendidos. Así pues, si cada punto es un bit, y una línea tiene ocho puntos, una línea de cualquier carácter se puede representar mediante ocho bits, es decir, con un byte. Dado que los caracteres completos están constituidos por ocho líneas, en la memoria podemos representarlos por ocho bytes consecutivos.

Asi por ejemplo, el carácter ESPACIO está memorizado con los ocho bytes (64 bits) que van desde 15616 hasta 15623 (ambos inclusive, la ADMIRACION, está almacenada desde 15624 hasta 15631, etc... Cada ocho bytes consecutivos, a partir del comienzo del generador de caracteres, representan la forma de un carácter completo, visualizados en la pantalla como sus ocho líneas, una debajo de otra.

Sabiendo unas poquitas matemáticas (de eso el ordenador entiende bastante), conociendo el punto de inicio del generador de caracteres y el código del carácter a buscar, al ordenador no le es nada difícil localizar la dirección del generador de caracteres, a partir de la cual se haya la for-

A los gráficos predefinidos les corresponden los códigos comprendidos entre 128 y 143.



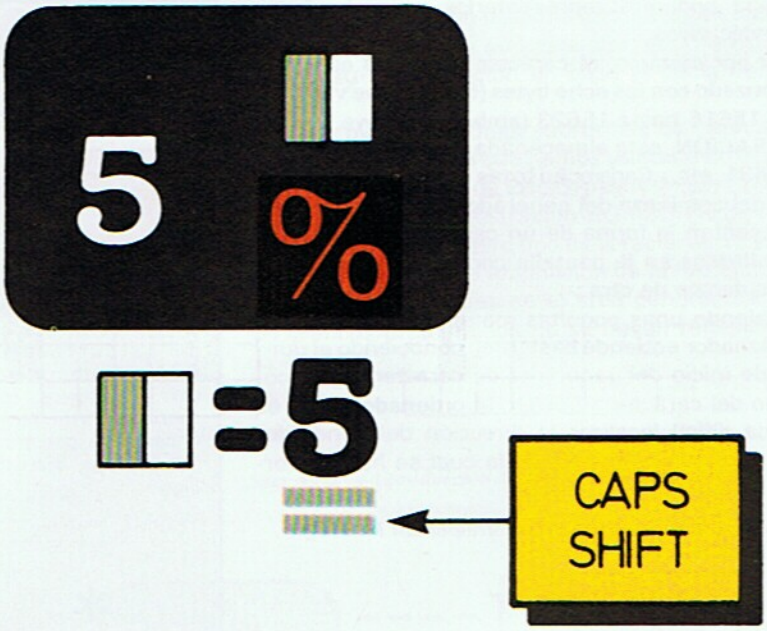
Los gráficos predefinidos no se encuentran en el área generadora de caracteres en ROM; siempre que son necesarios, una rutina del *firmware* se encarga de construirlos en base a su código.

*

Los caracteres que aparecen doblemente subrayados corresponden a los gráficos cambiados de las teclas cuyo carácter principal es el afectado por el subrayado. Se denominan gráficos cambiados a aquellos obtenidos pulsando la tecla correspondiente en combinación con **CAPS SHIFT**.

*

El área generadora de caracteres en ROM contiene las configuraciones de los caracteres entre los códigos 32 (espacio) y 127 (copyright).



Los caracteres subrayados doblemente corresponden a los gráficos cambiados de las teclas.

Como hemos visto hace tan solo unos instantes, las formas de los caracteres se almacenan en una zona de la memoria denominada generador de caracteres; sin embargo, no todos ellos se encuentran reflejados en ella, sino únicamente lo de códigos comprendidos entre 32 y 127. ¿Qué sucede con los restantes?

Los primeros treinta y dos (del 0 al 31), no suponen ningún problema, puesto que se trata de caracteres de control, sin forma determinada, y que en caso de verse en la obligación de imprimirlos (por ejemplo, **PRINT CHR$ 0**), los sustituye por un signo de interrogación.

A partir del 165 se encuentran las TOKEN, que lógicamente no tienen representación en el generador de caracteres, puesto que se componen de varios caracteres simples. Del mismo modo, los gráficos predefinidos, que abarcan desde el código 128 hasta el 143 (ambos inclusive), son generados por el Spectrum cuando es necesaria su utilización, debido a lo cual tampoco están presentes en el área de caracteres. Ahora bien, para la definición de los gráficos del usuario, es de máximo interés que conozcamos su situación en la memoria.

Como es lógico, no se encuentran dentro del generador de caracteres, puesto que ello simplicaría que se hallaran en la ROM, y por tanto, que

¡!

Se denominan TOKEN a las palabras BASIC multicarácter, las cuales ocupan los últimos 128 códigos del juego A.S.C.I.I. del Spectrum.

*

Los bits a uno en un carácter, corresponden a los puntos encendidos de la pantalla, es decir, los que se presentan en el color de primer término (INK).

ma del carácter a representar: **Dirección=15616+(Código-32)*8;** para todo código entre 32 y 127.

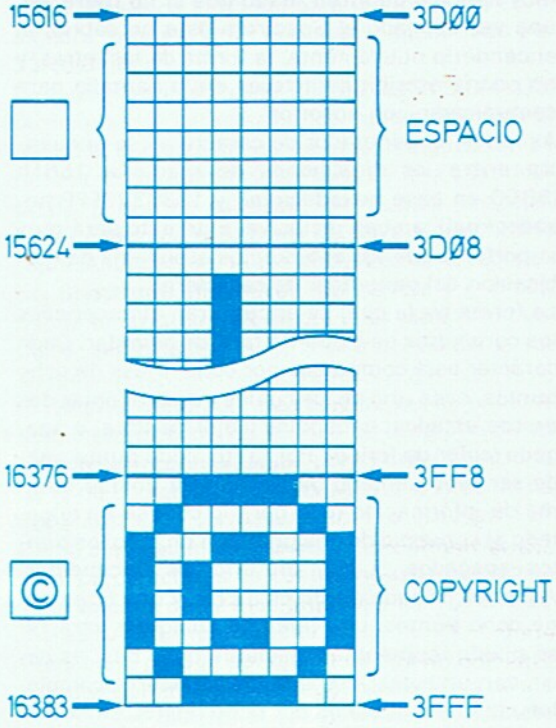
CARACTERES GRAFICOS DEFINIBLES

Además de los dieciséis caracteres gráficos predefinidos, el Spectrum tiene veintiuno más disponibles, para definirlos con la configuración exacta que decidamos. Para acceder a ellos, basta con situar el cursor en el modo gráfico (G), y pulsar una tecla de la A a la U.

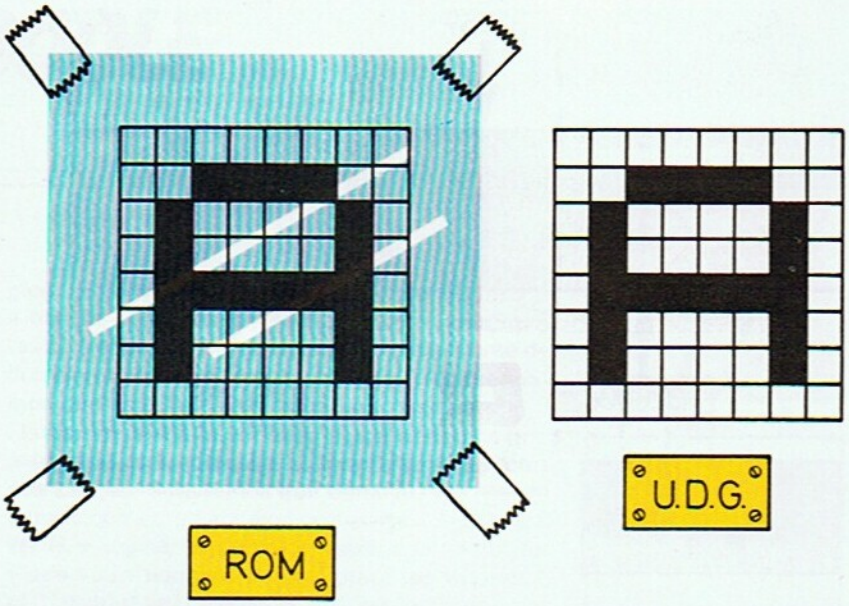
Si antes de efectuar ninguna definición de caracteres, pretendemos acceder a un gráfico definible, nos encontraremos con que su configuración es exactamente la misma que la correspondiente al carácter alfabético que representa.

Esto es debido a que el Spectrum supone, como valores por defecto para cada una de las teclas de los gráficos definibles, el carácter principal que representa su tecla. Ahora veremos por qué.

El generador de caracteres en ROM comienza en la dirección 15616 y acaba en 16383.



su configuración fuera inalterable, lo cual va contra su propia esencia. Así pues, estos se deben localizar en algún lugar de la memoria RAM (disponible para el usuario).
Los 168 bytes que ocupan (21 gráficos por ocho bytes cada uno), se encuentran situados concretamente al final de la memoria. En el modelo de 16 K a partir de la dirección 32600 (hasta 32767), y en el de 48 K o Plus a partir de 65368 (hasta 65535). Como ya hemos anticipado, la forma en la cual se almacenan estos caracteres dentro de su zona es la misma que la de los caracteres predefinidos en el generador en ROM: 8 bytes consecutivos por carácter.
Como es lógico, al estar estos caracteres ubicados en la RAM, cuando se enciende el ordenador no tienen ningún valor inicial (la RAM se borra al desconectar el Spectrum de la corriente). Por ello, entre las tareas que el ordenador lleva a cabo nada más conectarse, antes de hacer aparecer el mensaje de presentación, se cuenta la de dar una forma inicial a los gráficos definibles, copiando el generador de caracteres desde la A hasta la U, aunque podremos alterar estos valores iniciales sin ningún problema.
Así pues, para la definición de los caracteres nos apoyaremos fundamentalmente en dos puntos, la obtención de los bytes que configuran el carácter tal como lo diseñamos, y la modificación del área de memoria que los alberga. A tal fin, el BASIC dispone de dos sentencias especiales que nos facilitan este trabajo: **BIN** y **POKE**.

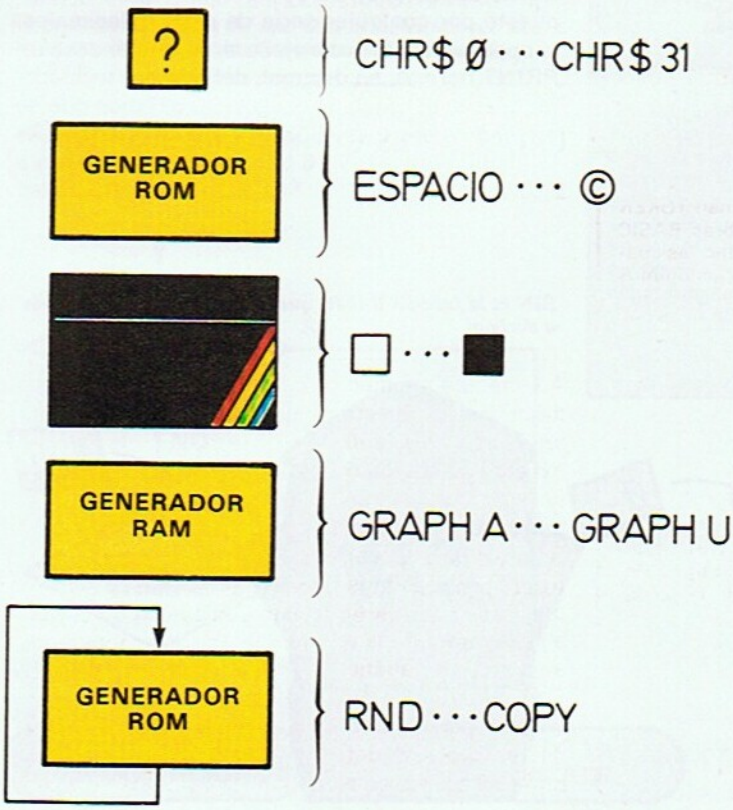


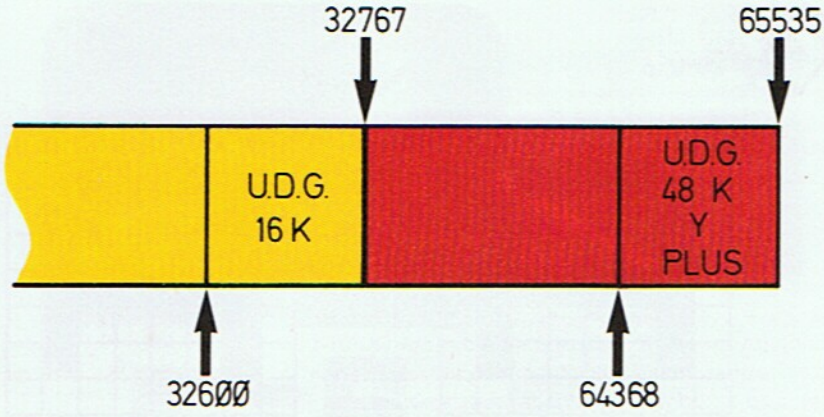
Aunque en un principio los UDG tengan la misma configuración que los caracteres de la ROM, pueden ser alterados con toda libertad.

BIN

El empleo de la función **BIN** permite introducir directamente en el ordenador números en formato binario, en vez de en la forma habitual (decimal). Como la definición del carácter abarca 8 líneas, será necesario teclear la configuración binaria de todas ellas, para definirlo completamente.
Cada una de las posiciones de memoria (líneas del carácter), puede adoptar valores entre 00000000 y 11111111 binario, cuya correspondencia en decimal es 0 y 255, respectivamente. Está claro que la notación es más conveniente para la definición de caracteres, puesto que se adapta mucho mejor a la forma en que se componen éstos. Sin embargo, conviene a veces, superada la fase de definición, incluir los datos de formación de los caracteres en los programas en el formato decimal, dado que éste resulta más corto y fácil de recordar.

Los caracteres del A.S.C.I.I. del Spectrum se obtienen de muy diversas maneras.





Las áreas de UDG (gráficos definibles por el usuario) se encuentran en diferentes lugares en el modelo de 16 y 48 K o Plus.

Los bits a cero en un carácter, corresponden a los puntos apagados de la pantalla, es decir, los que se presentan en el color de fondo (**PAPER**).

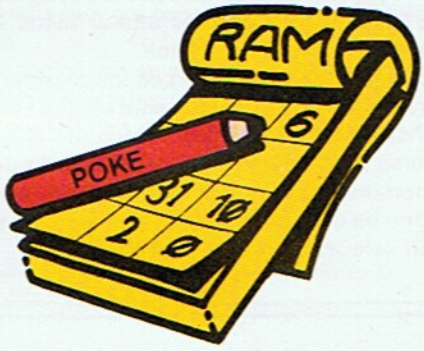
Un carácter viene definido en la memoria por una serie de ocho bytes consecutivos, cada uno de los cuales representa una línea del mismo. Dentro de cada byte (línea), los bits indican el estado de un punto de la pantalla.

El formato de la función **BIN** es sencillo. Consta de la propia palabra y un argumento de 1 a 16 dígitos binarios (deben ser forzosamente unos o ceros); lógicamente, como toda función BASIC, **BIN** habrá de ir precedida de una palabra clave. **PRINT BIN** realiza la conversión del argumento suministrado en binario, a una respuesta impresa en la pantalla en decimal. Cuando no se expresa ningún argumento, el resultado de la conversión es 0. Por otra parte, cuando como argumento suministramos un número negativo, compuesto por cualquier serie de dígitos decimales, la operación **BIN** no se ejecuta, produciéndose un **PRINT** normal, en decimal, del número tecleado. En cuanto al rango de operación, diremos que se encuentra entre 0 y 65535 decimal, o lo que es lo mismo, entre 0 y 16 bits, que corresponden a la ocupación de dos posiciones de memoria consecutivas, aunque para el caso de la definición de caracteres, sólo nos será necesario hacer uso de números binarios de 1 a 8 dígitos.

BIN *es la función BASIC que permite traducir de binario a decimal.*

POKE

Esta sentencia deposita un valor numérico entero entre 0 y 255, en la posición de memoria que deseemos (de 0 a 65535). Por tanto, modifica el contenido de la memoria en cualquier punto de la misma, depositando un nuevo byte.
La dirección de memoria sobre la cual ejerce su acción **POKE** puede pertenecer a la ROM, puesto que no se producirá ningún error de ejecución, pero carecerá de efecto alguno. Si se facilita una dirección fuera de límites, es decir, menor que cero o mayor que 65535, obtendremos el error **B Integer out of range** (entero fuera de rango). Sin embargo, la utilización de direcciones con decimales sí que está permitida, aunque el ordenador se encarga antes de depositar el byte, de redondear al entero más próximo la dirección facilitada.



POKE *deposita un valor de la memoria.*

En cuanto al rango en el cual se mueve el byte a depositar, se encuentra entre –255 y 255, obteniéndose el error **B Integer out of range** al emplear valores fuera de este límite. Como en el caso de las direcciones, los valores decimales se redondean al entero más próximo: desde 0.5 en adelante, al superior, y hasta 0.49..., al inferior. Por último, los valores negativos equivalen al complemento hasta 255, es decir, –1 equivale a 255, –2 a 254, etc... hasta –255 que equivale a 1.

ALGEBRA DE BOOLE

YA en el capítulo anterior comenzamos nuestra introducción al tema del álgebra de Boole, observando el funcionamiento de la primera de las operaciones lógicas: AND.

A continuación pasaremos al estudio pormenorizado del resto de las funciones lógicas, lo cual nos brindará la base suficiente para comprender la relación de este área de las matemáticas con la informática, y más concretamente con la circuitería (*hardware*) de nuestro Spectrum. Pasemos por tanto a la siguiente operación lógica.

OR

Esta operación se representa mediante un signo más (+), pero debemos de tener mucho cuidado para no confundirla con la operación aritmética suma. Su tabla de verdad es la siguiente:

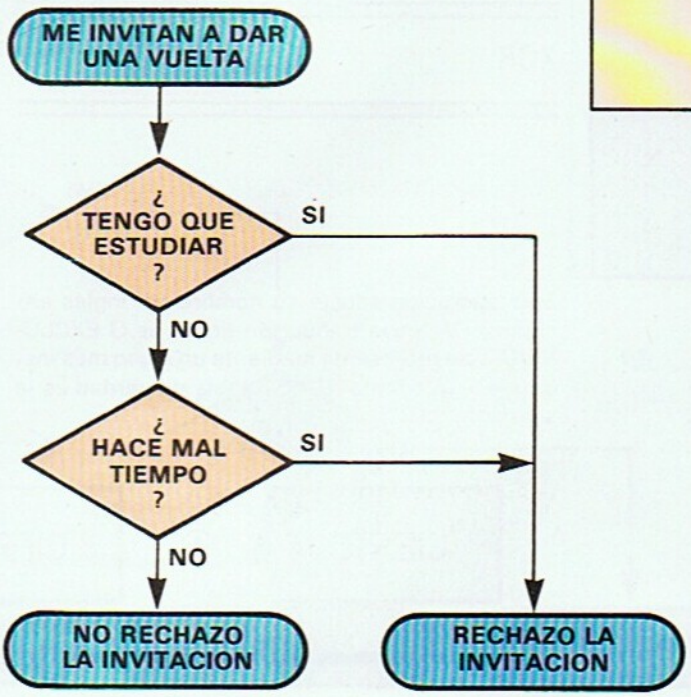
A + B	Q
0 + 0	0
1 + 0	1
0 + 1	1
1 + 1	1

Dada la similitud de los resultados obtenidos por esta función con los de una suma aritmética, en el caso de que los valores fueran considerados como cuantificables (números) y no como valores lógicos, esta operación se conoce también bajo la denominación de SUMA LOGICA.

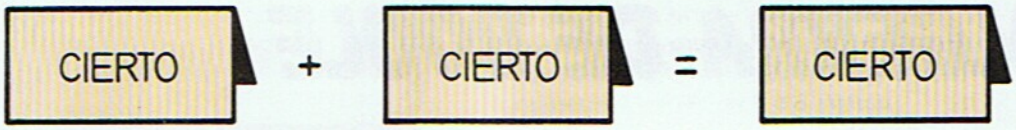
El nombre de OR, palabra inglesa cuyo significado es O, se asocia con el tipo de problemas lógicos sobre los cuales se aplica: SI <condición> O <condición> ENTONCES ... <resultado>. Así pues, si con AND es indispensable que sean ciertas ambas condiciones para que sea cierto el resultado, con OR es suficiente que cualquiera de las dos se cumpla, aunque no plantea ningún problema que las dos condiciones se cumplan. Veamos un ejemplo de este tipo de estructura lógica. Supongamos que un amigo nos propone ir a dar un paseo; el que rechacemos o no la invitación (resultado lógico) depende directamente de dos circunstancias (variables lógicas): que tengamos que estudiar y que haga buen tiempo.

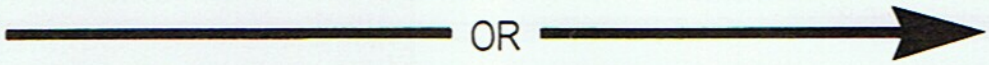
Llamaremos A a la variable lógica «tenemos que estudiar», que como es habitual puede adoptar dos valores: 1 (tenemos que estudiar) y 0 (no tenemos que estudiar). Por otra parte, la segunda variable lógica, B, corresponderá a la condición «hace buen tiempo», que adoptará los valores 1 o 0 según se cumpla o no, respectivamente. Al resultado de nuestro problema le denominaremos Q, equivaliendo a «rechazamos la invitación»; como es normal, el resultado podrá también adoptar dos únicos valores; 1 si rechazamos la invitación y 0 si no lo hacemos. Dado que es suficiente que cualquiera de las dos variables se cumpla para verificar el resultado, nos encontra-

En la estructura lógica del tipo OR, es suficiente que una de las condiciones se cumpla para que se verifique el resultado.



La operación lógica AND se representa por un círculo relleno (●), OR por un signo más (+), XOR por un signo más inscrito en un círculo (⊕) y NOT por una raya horizontal sobre el operando (−).





Para que se verifique un resultado de la operación XOR, sus operandos deben encontrarse en estados diferentes.

NOT es una operación lógica con un solo operando, cuya misión es cambiar el estado de éste.
V

mos ante una operación OR. Sólo nos resta asignar los valores para cada caso, y hacer las correspondientes consultas en la tabla de verdad. Si no tenemos que estudiar (A=0) y no hace mal tiempo (B=0) no rechazaremos la invitación (Q=0), puesto que 0 + 0 = 0. Si tenemos que estudiar (A=1) y no hace mal tiempo (B=0) rechazaremos la invitación (Q=1), dado que tenemos que quedarnos en casa para estudiar: 1 + 0 = 1.

Algo muy parecido ocurre en el siguiente caso: si no tenemos que estudiar (A=0) pero hace mal tiempo (B=1) rechazaremos la invitación (Q=1), puesto que 0 + 1 = 1. Por último, si ambas condiciones se cumplen, es decir, si tenemos que estudiar (A=1) y además hace mal tiempo (B=1), evidentemente rechazaremos la invitación (Q=1); efectivamente, 1 + 1 = 1. No olvidemos que el signo más (+) indica en estos casos una suma lógica y no una aritmética.

Sin duda las dos operaciones lógicas más comunes son **AND** y **OR**, sobre todo para los usuarios del BASIC, puesto que disponemos de dos operadores lógicos, con los mismos nombres y misiones que los aquí estudiados, los cuales nos permiten obtener este tipo de estructuras lógicas en combinación con la sentencia de decisión **IF**.

La operación OR es también conocida como SUMA LOGICA.

XOR

Esta operación adopta su nombre del inglés eXclusive OR, cuya traducción literal es O EXCLUSIVO, y se representa mediante un signo más inscrito en un círculo (⊕). Su tabla de verdad es la siguiente:

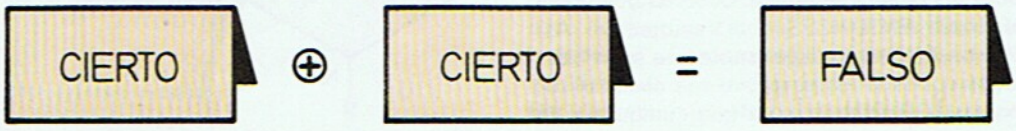
A⊕B	Q
0⊕0	0
1⊕0	1
0⊕1	1
1⊕1	0

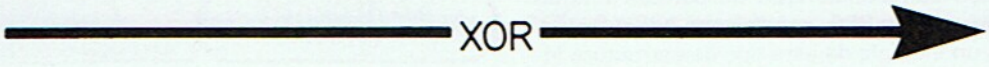
Constituye por tanto una estructura en la cual es preciso que sólo una de las condiciones se cumpla para que el resultado se verifique (sea 1). Con un ejemplo comprenderemos mejor su significado. Tenemos dos buenos amigos: Juan y Pedro; con cada uno de ellos nos llevamos a las mil maravillas, pero sin embargo, ellos no se aprecian, y siempre que se ven comienzan a discutir. Nos avisan unos amigos de que van a salir a dar un paseo y nos proponen acompañarles.

Puesto que hemos terminado los exámenes y hace buen tiempo, el que aceptemos su invitación (resultado lógico) depende de dos factores: la asistencia de Pedro (variable lógica A) y la asistencia de Juan (variable lógica B). Pero de una manera un tanto especial.

Si no van ni Pedro ni Juan (A=0) y B=0), no aceptaremos la invitación (Q=0), puesto que nos aburriremos (0 + 0 = 0). Si va Pedro (A=1) y no va Juan (B=0), no rechazaremos la invitación (Q=1): Pedro es un buen amigo y nos divertiremos (1 ⊕ 0 = 0). Análogamente, si no asiste Pedro (A=0) pero sí Juan (B=1), aceptaremos la invitación

En la tabla de verdad de la operación XOR, sólo dan como resultado uno los bits en diferente estado.





(Q=1), dado que tampoco nos aburriremos (0 ⊕ 1 = 1). Por último, si van tanto Pedro (A=1) como Juan (B=1), la mezcla se hace explosiva y el disgusto seguro, debido a lo cual decidimos no ir con ellos (Q=0, puesto que 1 ⊕ 1 = 0). Así pues, hemos resuelto un problema lógico del tipo XOR.

NOT

Esta es la última de las funciones lógicas, y se diferencia de las demás en que actúa sobre un solo operando. Recibe su nombre (NOT en inglés significa NO) de la misión que le es encomendada, consistente en «negar» a su operando, es decir, cambiar su valor lógico, cualquiera que sea éste. Su representación es una raya horizontal sobre el valor negado. La tabla de verdad de esta operación, al tener un solo operando, es la más simple de todas:

$\overline{A}$	Q
$\overline{0}$	1
1	0

Esta función es la representación del más puro espíritu de la contradicción, y recuerda a los dos compañeros que no se llevan nada bien y se encuentran por la calle: «¿Hacia donde vas Andrés, hacia la derecha o hacia la izquierda?» Si Andrés va hacia la derecha, Manolo piensa ir hacia la izquierda, y si Andrés dice ir hacia la izquierda, Manolo se dirigirá a la derecha.
Con NOT finalizamos la revisión de las operaciones lógicas, y a continuación pasamos a analizar la aplicación de las mismas a la informática.

APLICACION DEL ALGEBRA DE BOOLE

Desde el punto de vista de la programación, la utilización habitual no es sobre un solo bit, tal como hemos visto en las tablas de verdad, sino sobre un byte. Este problema de las operaciones

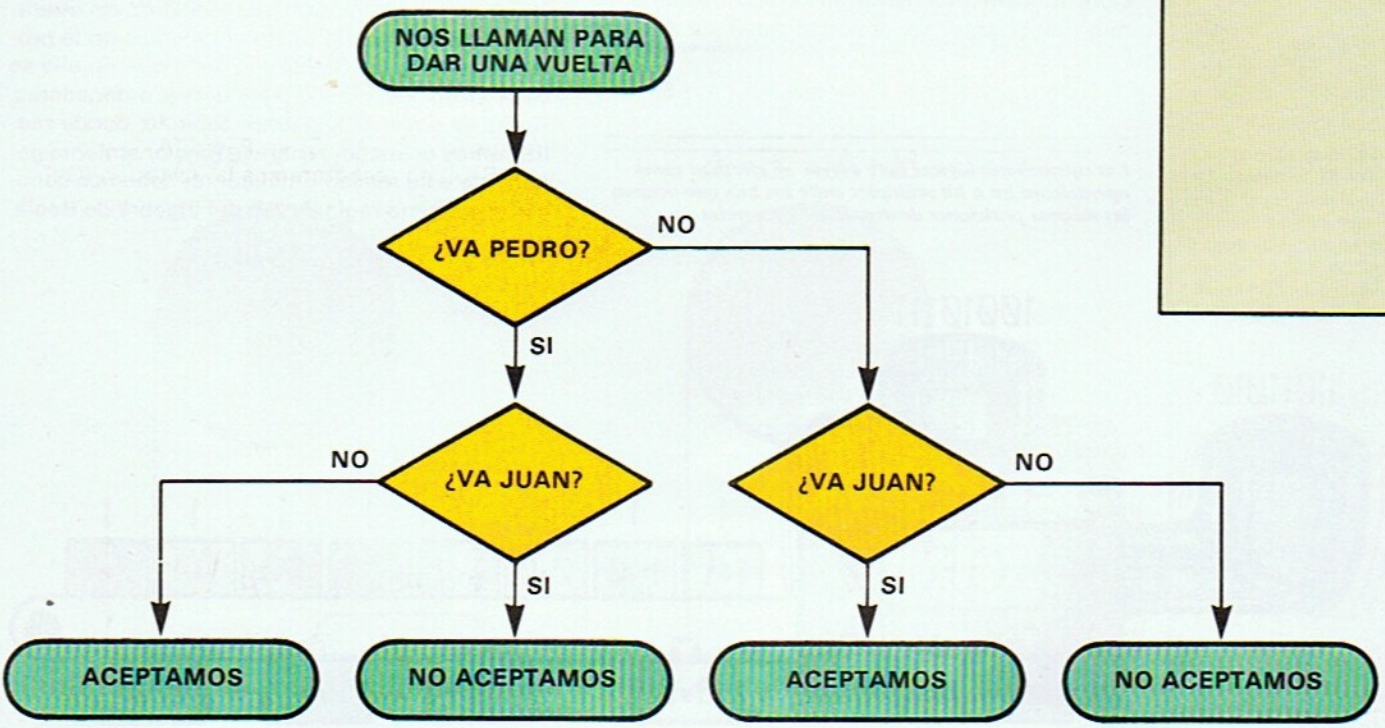
Para que se verifique el resultado en las estructuras XOR, es necesario que sólo una de las condiciones se cumpla.

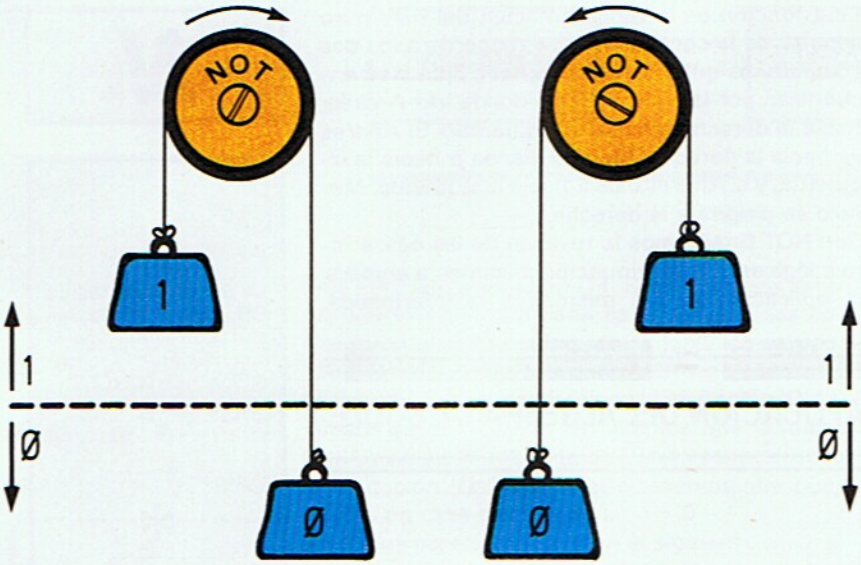
¡!

La operación lógica OR se utiliza en las estructuras del tipo Si <condición> O <condición> ENTONCES ... <resultados>.

*

En la tabla de verdad para OR, todos los resultados dan uno, salvo cuando ambas condiciones son cero.





La operación lógica NOT siempre cambia el estado de su operando.

multibit, se resuelve realizando las operaciones simples de los bits que ocupen iguales posiciones en los dos bytes a operar.

Así por ejemplo, el resultado de 01010100 • 10010001 es 00010000, puesto que, 0•1=0, 1•0=0, 0•0=0, 1•1=1, 0•0=0, 1•0=0, 0•0=0 y 0•1=0.

La aplicación de estas funciones a la programación, fundamentalmente en código máquina, es muy importante. La principal tarea de AND es anular determinados bits de un byte, dejando los otros intactos. Así por ejemplo, realizar un AND de cualquier operando con el byte 11110000, asegura que sea cual sea éste, el resultado corresponderá a sus cuatro bits de la izquierda inalterados y los restantes con valor cero.

OR por el contrario, se emplea para asegurarse que determinados bits del resultado sean uno, y que el resto permanezcan inalterados. Por ejemplo, realizar un OR de cualquier operando con el byte 11110000, asegura que los cuatro bits de la izquierda del resultado serán unos, y que los cuatro menos significativos (los de la derecha) no se verán afectados.

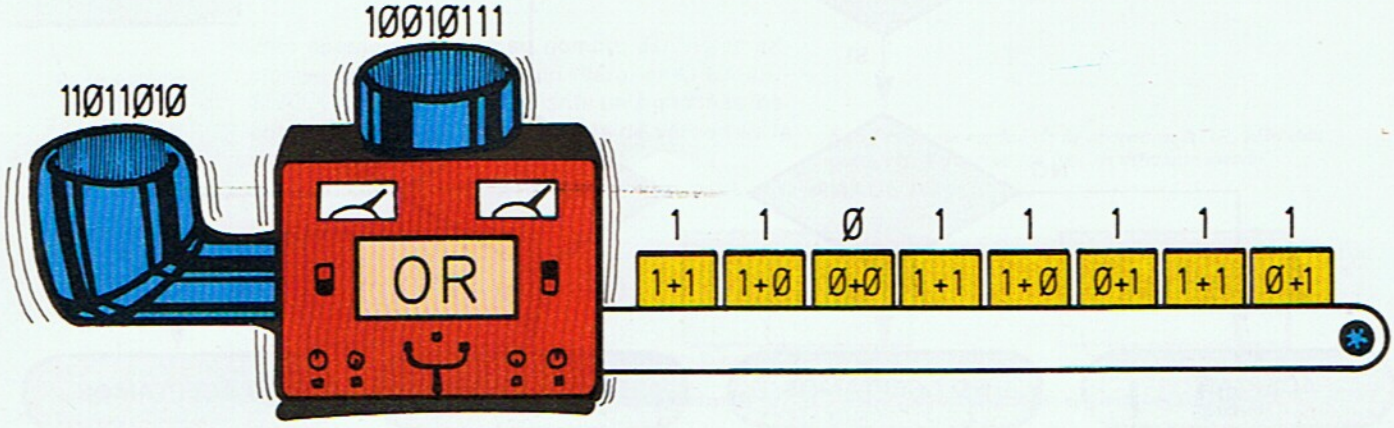
A estas técnicas basadas en AND y OR, consistentes en preparar uno de los operandos para que sea cual sea el otro, le anule o active determinados bits en el resultado dejando el resto intactos, se les denomina ENMASCARAMIENTOS.

XOR tiene la propiedad de que al efectuarlo dos veces consecutivas se retorna al operando inicial, es decir, A⊕B = Q y Q⊕B = A. Otra de sus propiedades es asegurar que el resultado será cero sean cuales sen A y B, siempre que se cumplan que son iguales: para todo A=B se cumple que A ⊕B = 0.

Por último, la misión de NOT es claramente obtener un resultado opuesto al del operando, cualquiera que sea éste. Esta función, hablando en términos de programación en código máquina, se suele conocer como complementación, y es de vital importancia en el sistema que se emplea para las operaciones aritméticas con bytes, en las cuales se tenga en cuenta el signo; la denominada aritmética en complemento a dos, que por el momento no es necesario que conozcamos.

Pero sin duda alguna, la aplicación más interesante del álgebra de Boole la tenemos en la propia circuitería del Spectrum, dado que en ella se basa el núcleo «inteligente» de los ordenadores. Antes de pasar al siguiente capítulo, donde realizaremos un estudio sobre el funcionamiento general de este núcleo «inteligente» debemos conocer una última regla de oro del álgebra de Boole.

Las operaciones lógicas byte a byte, se efectúan como operaciones bit a bit múltiples entre los bits que ocupan las mismas posiciones dentro de los operandos.



AND

1 0 1 0 1 0 1 0 • 1 1 1 1 0 0 0 0 = 1 0 1 0 0 0 0 0

OR

1 0 1 0 1 0 1 0 + 1 1 1 1 0 0 0 0 = 1 1 1 1 1 0 1 0

XOR

1 1 1 1 0 0 0 0 ⊕ 1 1 1 1 0 0 0 0 = 0 0 0 0 0 0 0 0

NOT

1 0 1 0 1 0 1 0 = 0 1 0 1 0 1 0 1

La principal utilidad de AND y OR es el enmascaramiento. XOR se utiliza frecuentemente para hacer cero un resultado, y NOT para la aritmética binaria en complemento a dos.

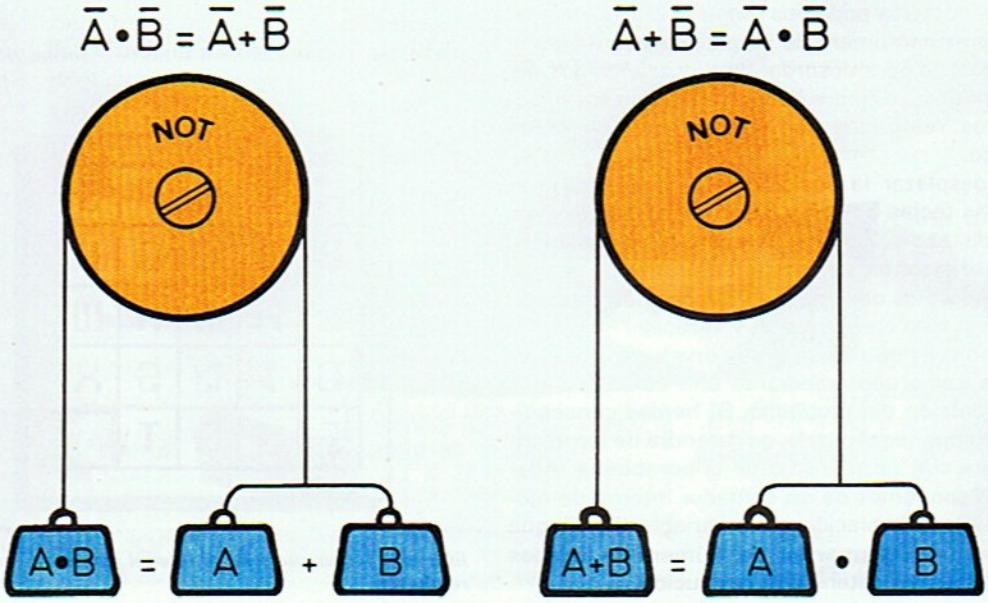
TEOREMA DE DE MORGAN

Un importante teorema en el álgebra de Boole es el de *De Morgan*, el cual puede ser formulado de dos formas diferentes: la negación de un AND es igual al OR de las negaciones, o bien, la negación de un OR es igual al AND de las negaciones. Esto se puede expresar mediante la simbología que ya conocemos de la siguiente manera.

$$\overline{A \bullet B} = \overline{A} + \overline{B}$$
$$\overline{A + B} = \overline{A} \bullet \overline{B}$$

De esta última norma hace uso muy frecuente el *hardware*, puesto que en determinadas circunstancias facilita el diseño de la circuitería del ordenador. Pero todo esto será explicado más profusamente en el siguiente capítulo.

El teorema de De Morgan *puede formularse de dos formas:*
$\overline{A \bullet B} = \overline{A} + \overline{B}$ *ó bien,* $\overline{A + B} = \overline{A} \bullet \overline{B}$



¡!

Las operaciones lógicas byte a byte, se efectúan como operaciones bit a bit múltiples, entre los bits que ocupen posiciones iguales en los operandos.

*

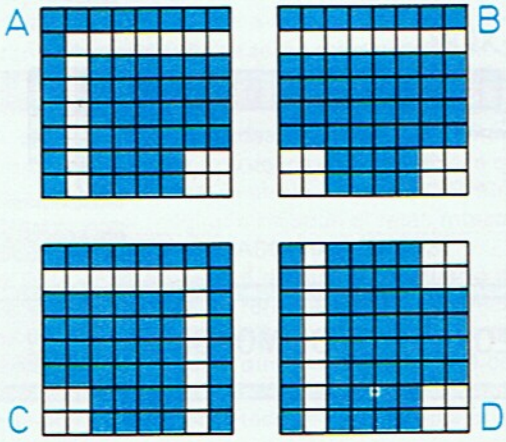
La técnica que permite asegurar la anulación o activación, de determinados bits del resultado en una operación lógica, se denomina ENMASCARAMIENTO.

*

Las dos posibles formas de enunciar el teorema de *De Morgan* son: $\overline{A \bullet B} = \overline{A} + \overline{B}$ y $\overline{A + B} = \overline{A} \bullet \overline{B}$.

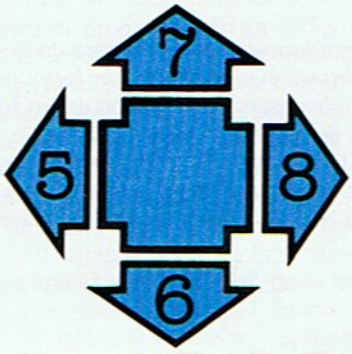
PUZZLE

NUESTRO programa Puzz es la versión informatizada de un clásico pasatiempo. El objetivo final de este juego es ordenar un alfabeto, encerrado dentro de un bastidor de 5 × 5 posiciones.
En el programa, dentro de este cuadrado, cada letra ocupa una posición de carácter, y una de las posiciones está desocupada. El juego nos permite desplazar el espacio en blanco a cualquier posición contigua, de forma que la letra que hubiera allí pase a ocupar la antigua situación del espacio. Gracias a esto, podemos ir desplazando las letras, para conseguir ordenar el alfabeto en el menor número posible de jugadas.
Cualquiera de las letras puede desplazarse a la posición en blanco, siempre que la situación de ésta y la del espacio sean adyacentes.



Caracteres gráficos para la definición del bastidor.

COMO JUGAR



Para el movimiento del espacio utilizaremos las teclas 5, 6, 7 y 8.

Al ejecutar el programa, aparecerá en la pantalla el alfabeto ordenado. Seguidamente, nuestro Spectrum se encargará de desajustar las posiciones originales de las letras. Este sistema asegura que nosotros podamos resolver el problema, al menos en el número de jugadas que la máquina ha utilizado para desordenar el puzzle, dado que, lógicamente, siempre podríamos seguir los movimientos realizados por ella, aunque en orden inverso.
Para desplazar la posición en blanco, utilizaremos las teclas 5, 6, 7 y 8 correspondientes a izquierda, abajo, arriba y derecha, respectivamente. Este sistema de movimiento implica que no nos podremos desplazar en direcciones diagonales, sino sólo en horizontal y vertical.
Por cada vez que efectuamos una jugada, el programa comprueba mediante una de sus rutinas la resolución del problema. Si hemos conseguido recomponer el puzzle, se detendrá de forma inmediata con un mensaje de enhorabuena. Además, disponemos de un contador interno de movimientos, establecido en la variable CONT, que le permitirá informarnos del número de jugadas en que hemos obtenido la resolución.

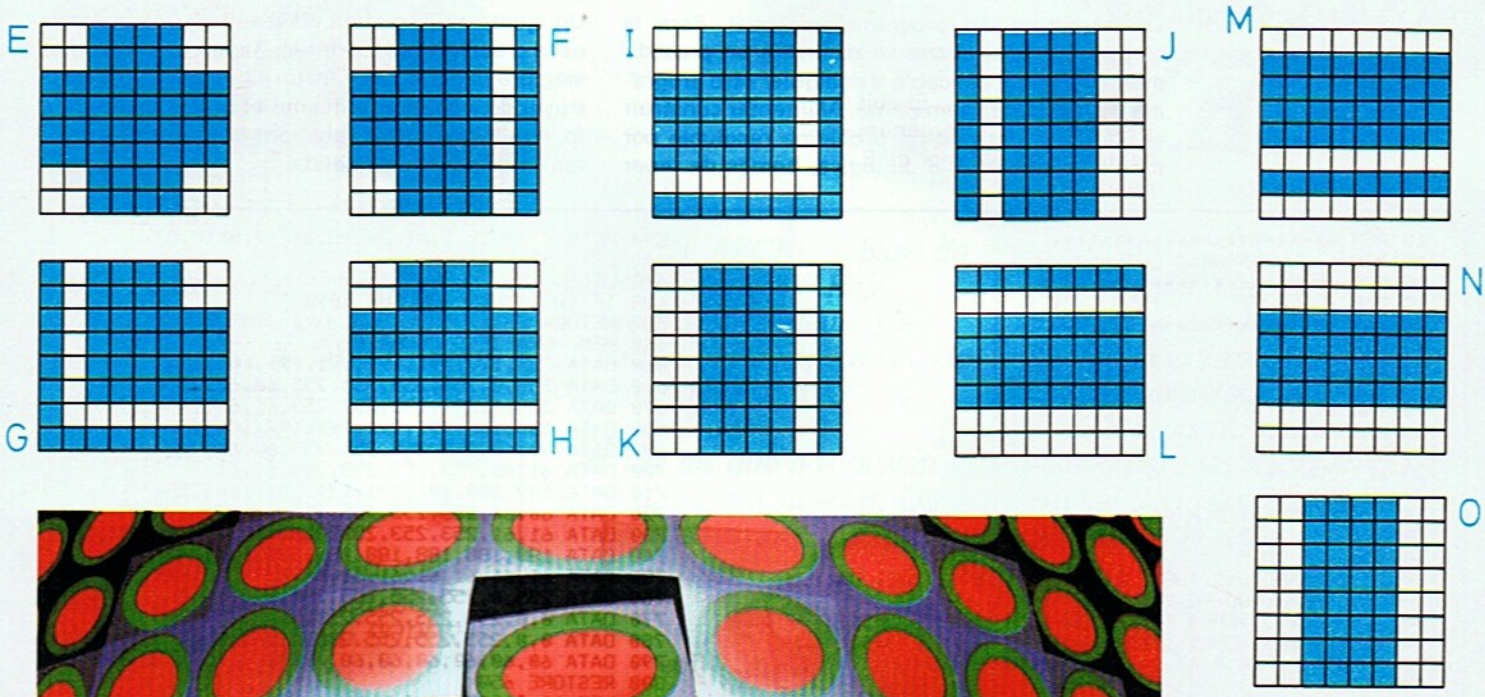
Esta es la forma que debe adoptar el puzzle tras su resolución.

¡!

El programa lo grabaremos usando el siguiente comando **SAVE "PUZZ"**. En el caso de que quisiéramos grabarlo con autoejecución, tecleríamos **SAVE "PUZZ" LINE 10**.

*

La introducción de gráficos implica el paso previo al cursor G (**CAPS SHIFT + 9**), para a continuación pulsar la tecla cuyo carácter principal es el subrayado.


E
F
I
J
M
G
H
K
L
N
O

EL PROGRAMA

La estructura del programa es lineal. Para la composición del puzzle se recurre a un procedimiento general aplicable a cualquier otro programa de naturaleza semejante. Al intentar construir el planteamiento de un problema resoluble por medio de habilidad o de lógica, hemos de tener en cuenta la condición primaria de que el problema ha de disponer de solución; esto es: hemos de construir la estructura inicial a partir de una solución preestablecida.

En este programa, se parte de una matriz ordenada y se descompone por medio de varios accesos a la subrutina de movimiento con valores aleatorios; de esta manera, queda asegurada la existencia de al menos una solución.

Un ejemplo típico de la adopción de este método es la creación de laberintos. Según este sistema, siempre hay que partir de un camino inicial, construyendo a partir del mismo el resto del laberinto, con lo cual queda siempre asegurada la existencia de esa vía de salida.

```
 10 REM ***********************
 20 REM * J.M.MAYORAL SERRANO *
 30 REM ***********************
 40 REM *   PUZZ      1985    *
 50 REM ***********************
 60 BORDER 0
 70 PAPER 0
 80 LET SW=0
 90 INK 9
100 RANDOMIZE
110 LET P=1: CLS
120 LET CONT=0
130 GO SUB 800
140 DIM D$(5,5): DIM M$(5,5)
150 FOR I=1 TO 5
160 FOR J=1 TO 5
170 IF I=5 AND J=5 THEN  LET M$(I,J)=" ": LET D$(I,J
)=" ": LET P=2: GO TO 200
180 LET M$(I,J)=CHR$ (I+5*J+59-32*(I=5 AND J=5))
190 LET D$(I,J)=CHR$ (I+5*J+59-32*(I=5 AND J=5))
200 PRINT  PAPER P;AT 2*I+5,2*J+9;M$(I,J)
210 NEXT J
220 NEXT I
230 LET F=5
240 LET FN=5
250 LET C=5
260 LET CN=5
270 PRINT AT 19,6; FLASH 1; PAPER 4;"   DESORDENANDO
!"
280 FOR I=1 TO 250
290 LET X$=STR$ (INT (RND*4)+5)
300 BEEP .01,40
310 GO SUB 460
320 NEXT I
330 PRINT AT 19,6; PAPER 0;"
340 LET SW=1
350 PRINT AT 21,0;"PULSA UNA TECLA PARA COMENZAR"
360 PAUSE 0
370 PRINT AT 21,0;"
380 GO SUB 1030
390 GO SUB 410
400 GO TO 390
410 IF INKEY$<>"" THEN  GO TO 410
420 IF INKEY$="" THEN  GO TO 420
430 LET X$=INKEY$
440 IF CODE X$<53 OR CODE X$>56 THEN  GO TO 410
450 LET CONT=CONT+1
460 IF X$="5" AND C=1 THEN  RETURN
470 IF X$="6" AND F=5 THEN  RETURN
480 IF X$="7" AND F=1 THEN  RETURN
490 IF X$="8" AND C=5 THEN  RETURN
500 BEEP .05,40
510 IF X$="5" THEN  LET CN=C-1
520 IF X$="6" THEN  LET FN=F+1
530 IF X$="7" THEN  LET FN=F-1
540 IF X$="8" THEN  LET CN=C+1
550 LET X$=M$(FN,CN)
560 LET M$(FN,CN)=M$(F,C)
570 LET M$(F,C)=X$
580 PRINT  PAPER 2;AT 2*FN+5,2*CN+9;M$(FN,CN)
590 PRINT  PAPER 1;AT 2*F+5,2*C+9;M$(F,C)
600 LET F=FN
610 LET C=CN
620 IF SW=1 THEN  GO SUB 1090
630 RETURN
640 REM  GRAFICOS USUARIO
650 DATA 255,128,191,191,191,191,188,188
660 DATA 255,0,255,255,255,255,60,60
670 DATA 255,1,253,253,253,253,61,61
680 DATA 188,188,191,191,191,191,188,188
690 DATA 60,60,255,255,255,255,60,60
700 DATA 61,61,253,253,253,253,61,61
710 DATA 188,188,191,191,191,191,128,255
720 DATA 60,60,255,255,255,255,0,255
730 DATA 61,61,253,253,253,253,1,255
740 DATA 188,188,188,188,188,188,188,188
750 DATA 61,61,61,61,61,61,61,61
760 DATA 255,0,255,255,255,255,0,0
770 DATA 0,0,255,255,255,255,0,255
780 DATA 0,0,255,255,255,255,0,0
790 DATA 60,60,60,60,60,60,60,60
800 RESTORE 650
810 LET A$="ABCDEFGHIJKLMNO"
820 FOR F=1 TO LEN A$
830 FOR K=0 TO 7
840 READ A
850 POKE USR A$(F)+K,A
860 NEXT K
870 NEXT F
880 REM  DIBUJO TABLERO
890 PAPER 2: INK 6
900 PRINT AT 6,10;"ALBLBLBLBLC"
910 PRINT AT 7,10;"J O O O O K"
920 PRINT AT 8,10;"DNENENENENF"
930 PRINT AT 9,10;"J O O O O K"
940 PRINT AT 10,10; "DNENENENENF"
950 PRINT AT 11,10; "J O O O O K"
960 PRINT AT 12,10; "DNENENENENF"
970 PRINT AT 13,10; "J O O O O K"
980 PRINT AT 14,10; "DNENENENENF"
990 PRINT AT 15,10; "J O O O O K"
1000 PRINT AT 16,10; "GMHMHMHMHMI"
1010 PAPER 0: INK 9
1020 RETURN
1030 REM  INDICA MOVIMIENTOS
1040 PRINT AT 0,0; PAPER 5;" 5"; INK 9;" IZQUDA. "
1050 PRINT AT 3,0; PAPER 2;" 8"; INK 9;" DERECHA "
1060 PRINT AT 0,20; PAPER 7;" 7 "; INK 9;" ARRIBA "
1070 PRINT AT 3,20; PAPER 6;" 6 "; INK 9;" ABAJO  "
1080 RETURN
1090 REM  COMPROBACION
1100 FOR K=1 TO 5
1110 FOR Q=1 TO 5
1120 IF D$(Q,K)<>M$(Q,K) THEN  RETURN
1130 NEXT Q
1140 NEXT K
1150 PRINT AT 19,8;"LO CONSEGUISTE EN"
1160 PRINT AT 21,10;CONT;"  JUGADAS"
1170 IF INKEY$="" THEN  GO TO 1170
1180 RUN
```

U.D.G.

ESTA es la abreviatura inglesa de *User Definable Graphics* (Gráficos Definibles por el Usuario). Bajo este nombre se encuadran los veintiún caracteres gráficos del Spectrum, comprendidos entre los códigos 144 y 164, ambos inclusive, cuya forma puede ser seleccionada por nosotros mismos.

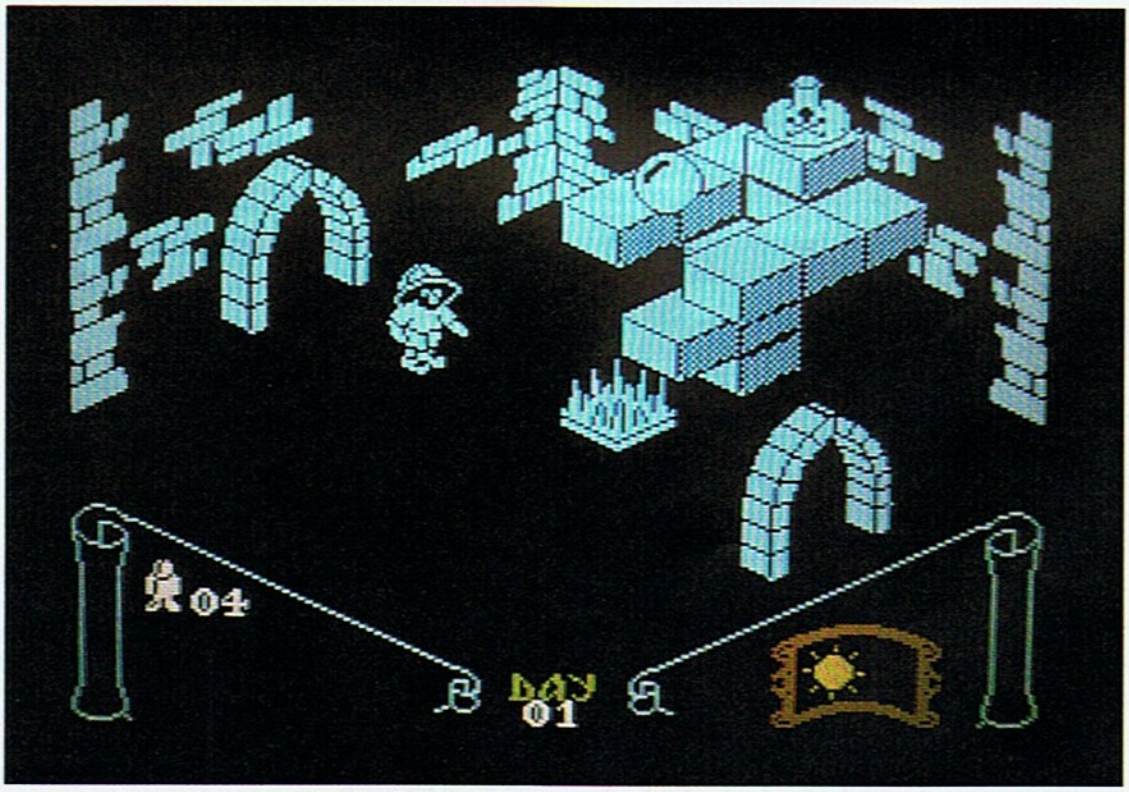
La definición de los caracteres gráficos no es una tarea que implique una gran complicación; se puede decir incluso que es bastante simple, si bien es cierto que si deseamos obtener resultados de una cierta calidad, requiere emplear una buena cantidad de tiempo en la labor de diseño. Estudiemos el proceso de generación de un gráfico, paso a paso.

La definición de caracteres es una tarea bastante simple, aunque precisa mucho tiempo para alcanzar resultados satisfactorios.

APARECE EL PROBLEMA

Nos encontramos ante el proyecto de llevar a cabo un programa educativo, que al estar destinado a la juventud, debe cuidar en su máximo extremo la ortografía y legibilidad de sus textos. Así pues, y dado que el Spectrum carece de algunos signos ortográficos castellanos, como las vocales acentuadas, la u con diéresis, etc., nos vemos en la obligación de buscar una solución que nos permita, de la forma más elegante, cómoda y rápida posible, obtener dichos caracteres.

En primer lugar, es necesario que conozcamos



¡!

U.D.G. es la abreviatura inglesa de *User Definable Graphics*: Gráficos Definibles por el usuario.

*

Aunque la definición de los caracteres no es una tarea muy complicada, sí precisa de bastante tiempo para la obtención de resultados de cierta calidad.

*

La obtención de caracteres especiales se puede conseguir por dos sistemas: combinación de caracteres predefinidos o generación de gráficos definibles.

¡!

El proceso de generación de un gráfico comienza con el diseño del mismo. En esta fase nos pueden resultar de gran ayu : los programas generadores de caracteres incluidos en las cintas de demostración de Sinclair.

*

Los ceros a la izquierda no tienen efecto alguno en el sistema de numeración binario.

exactamente la magnitud de nuestro problema, es decir, los caracteres concretos que nos van a ser necesarios:

— las cinco vocales acentuadas
— la u con diéresis
— la letra eñe, tanto mayúscula como minúscula
— el signo de apertura de interrogación
— el signo de apertura de admiración

Una vez hecho esto, debemos proceder a discernir cual es el método por el cual intentaremos conseguir nuestro objetivo.

Hace bien poco, pudimos ver como mediante la función **OVER**, nos era posible obtener tanto las vocales acentuadas, como la eñe minúscula, ahorrándonos el trabajo de la definición de caracteres, si bien es cierto que con un resultado bastante menos estético. En nuestro caso concreto, queda descartado este sistema, pues no nos posibilita la obtención de algunos de los caracteres que nos son indispensables.

Teniendo que proceder a la definición de caracteres, ¿tenemos suficiente capacidad con los U.D.G., o será necesaria una reubicación del juego de caracteres? Aunque aún no alcancemos a comprender la diferencia entre estos dos sistemas, diremos que si bien es más sencilla la definición de U.D.G., estos son sólo veintiuno, y en algunos casos serán insuficientes.

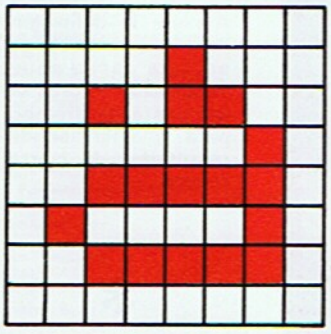
El Spectrum carece de ciertos caracteres o signos de uso frecuente en el castellano (vocales acentuadas, eñe, etc...).

Por otra parte, cuando al final de este capítulo estudiemos el sistema de reubicación del juego de caracteres, veremos que su manejo es ligeramente más complicado, y sobre todo, ocupa bastante más memoria.

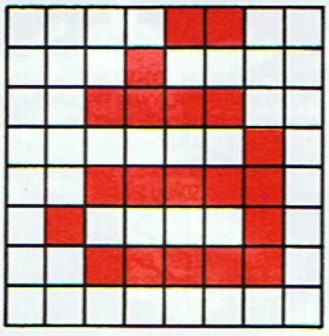
Así pues, dado que el número de caracteres que precisamos se limita a diez, tenemos la suficiente capacidad en los U.D.G. sin necesidad de recurrir a mayores complicaciones. Una vez decidido el sistema de obtención, llega la fase de definición de los caracteres.

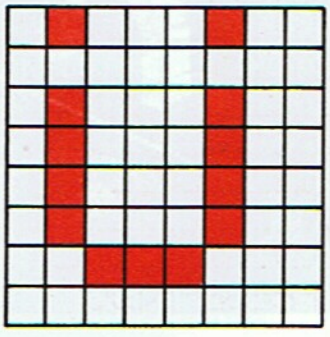
Aunque algunos caracteres especiales se pueden conseguir por combinación de predefinidos, otros están vedados a este sistema.

PREDEFINIDO



DEFINIDO





DISEÑO DE CARACTERES GRAFICOS

Para diseñar un carácter gráfico, basta con que nos hagamos con una hoja de papel cuadriculado, y marquemos sobre ella una zona de ocho por ocho cuadrículas.

Hecho esto, debemos dibujar la silueta del carácter en cuestión, con el símbolo que deseamos. Aunque el diseño del carácter así obtenido puede parecer, en principio, bastante tosco, no debe preocuparnos, puesto que la ampliación proporcionada por el papel cuadriculado es bastante grande, y su aspecto parecerá mucho más uniforme cuando se vea representado a su tamaño normal en la pantalla.

Concluida la operación de diseño, que sin duda es la más ardua hasta obtener un resultado lo suficientemente satisfactorio, el aspecto de nuestro dibujo debe ser algo similar a un tablero de ajedrez muy particular, en el cual los puntos ocupados tomarán el color del atributo **INK**, y los que permanecen libres el del **PAPER**.

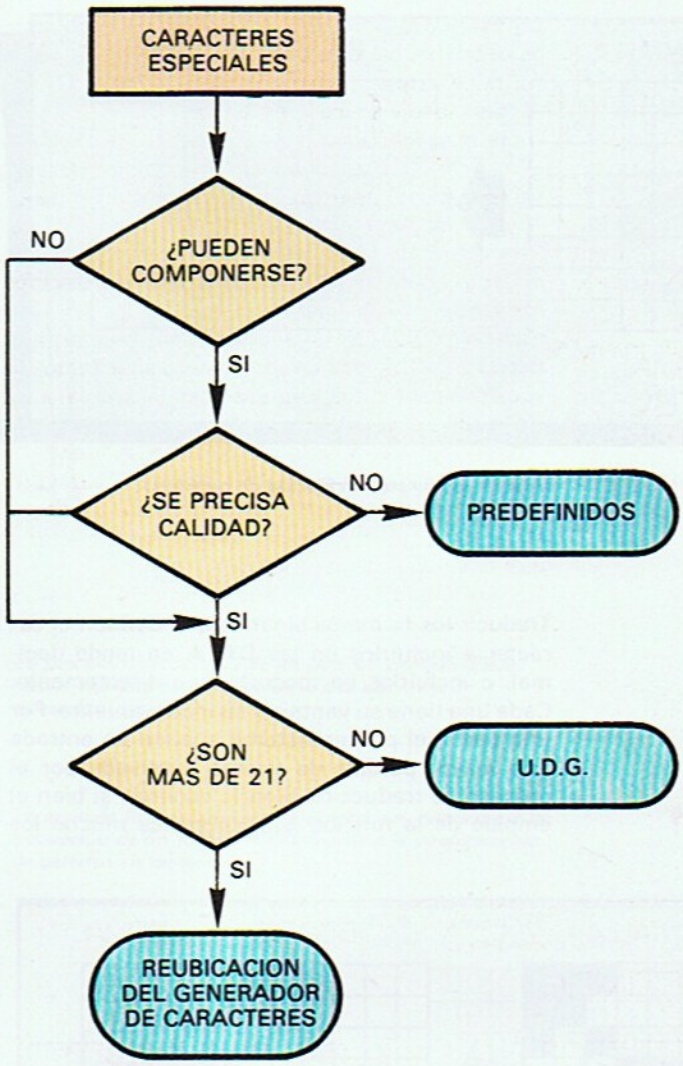
La tarea que debemos emprender a continuación es la de traducir nuestro carácter a un formato inteligible por el Spectrum. Puesto que nuestro ordenador, como todos los de su clase, sólo entiende de «unos» y «ceros», es decir, el lenguaje binario, debemos codificarle el carácter de esta forma.

Esto es mucho más fácil de lo que parece, puesto que se limita a sustituir los puntos coloreados de la cuadrícula por «unos» y los que permanecen en blanco por «ceros». De esta forma, cada una de las ocho líneas que componen el carácter se mostrará como una sucesión de 8 elementos, con valores 1 ó 0.

Así por ejemplo, una «a» acentuada podría tener el siguiente formato:

00001100
00010000
01111100
00000010
00111110
01000010
00111111
00000000

En este proceso, nos pueden servir de ayuda los denominados PROGRAMAS GENERADORES DE CARACTERES. El fabricante de nuestro aparato, consciente de la importancia de esta faceta, ha incorporado en sus cintas de demostración, tanto del ZX Spectrum como del Plus, un programa de este tipo, que no ayudará considerablemente en la fase de diseño.

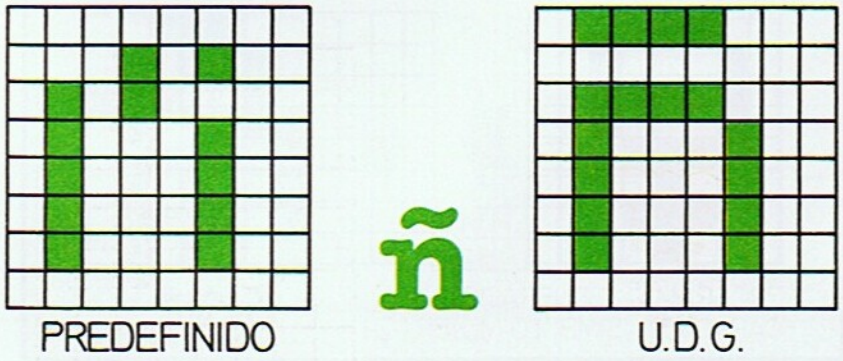


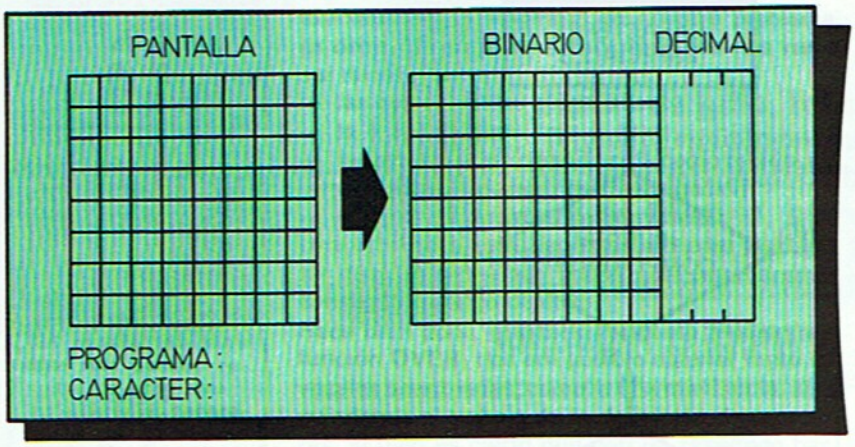
La elección de un método de obtención de caracteres especiales, se basa fundamentalmente en la calidad que se espere obtener, y la cantidad de ellos.

DEFINIENDO CARACTERES

La definición de un carácter concreto, nos da paso a la obtención de símbolos que no podemos encontrar en el teclado. El sistema más cómodo para incorporar los datos de los caracteres definidos a un programa es mediante sentencias **DATA**. Así pues, dado que la función BASIC **BIN** nos permite traducir números binarios a decimales, podremos optar por dos soluciones:

Aun en el caso de poder utilizar combinaciones de caracteres predefinidos, es considerablemente mayor la calidad obtenida con los U.D.G.





Hoja de diseño para definición de caracteres.

Traducir los formatos binarios que definen el carácter e incluirlos en las **DATA** en modo decimal, o incluirlos en modo binario directamente. Cada una tiene su ventaja y su inconveniente. Por una parte, el primer sistema supone de entrada una mayor pérdida de tiempo, motivada por el proceso de traducción binario decimal, si bien el empleo de la función **BIN** simplifica mucho los trámites. Por otra parte, dado que la ocupación del byte en binario es siempre mayor que en decimal, la inclusión de las **DATA** con este formato hace más pesada la labor de introducción del programa, y consume mayor cantidad de memoria.

Nosotros optaremos en nuestros ejemplos por el formato decimal; recurrimos a la estructura **PRINT BIN...** para ir averiguando los valores decimales de las líneas. Siguiendo con el ejemplo anterior:

```
PRINT BIN 00001100
PRINT BIN 00010000
PRINT BIN 01111100
PRINT BIN 00000010
PRINT BIN 00111110
PRINT BIN 01000010
PRINT BIN 00111111
PRINT BIN 00000000
```

Aunque no tengan efecto alguno, añadimos los ceros a la izquierda para representar todos los puntos del carácter. Los resultados de las operaciones del ejemplo serán, respectivamente: 12, 16, 124, 2, 62, 66, 63 y 0.

Una vez incluidas las sentencias **DATA** en el programa, es necesario conocer la forma de ubicar la definición del carácter. Para ello, debemos recurrir a una combinación de funciones del BASIC muy específica, denominada **POKE USR**, que adopta la forma general:

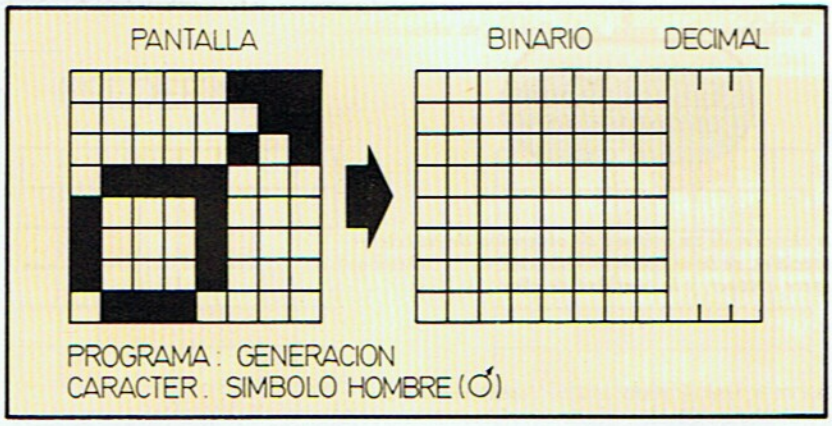
```
POKE USR "X",Y
```

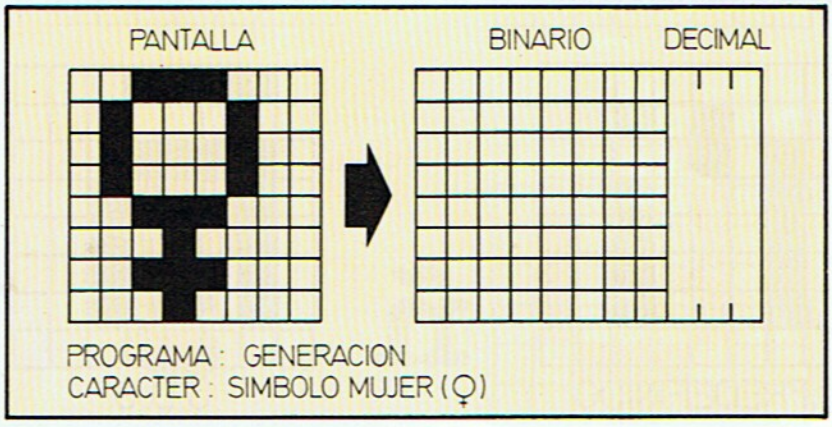
Donde **"X"** indica el carácter gráfico que se desea generar, e **Y** la configuración decimal de cada una de las ocho líneas que definen el carácter, o la binaria precedida de la función **BIN**.

Realmente, el conjunto **USR "X"** (siendo X un carácter de la A a la U), nos ofrece la posición de comienzo en memoria del gráfico correspondiente a la letra. Por lo tanto, en esa posición y las 7 siguientes, deben efectuarse las asignaciones de valores de cada una de las 8 líneas que componen cada carácter gráfico.

Estas asignaciones, tienen lugar por medio de la sentencia **POKE**, que coloca en la posición señalada por el primer parámetro (**USR "X"**) el valor del segundo (**Y**).

Veamos a continuación un ejemplo de ello en el





La primera fase del proceso de generación de gráficos, corresponde al diseño de los caracteres necesarios.

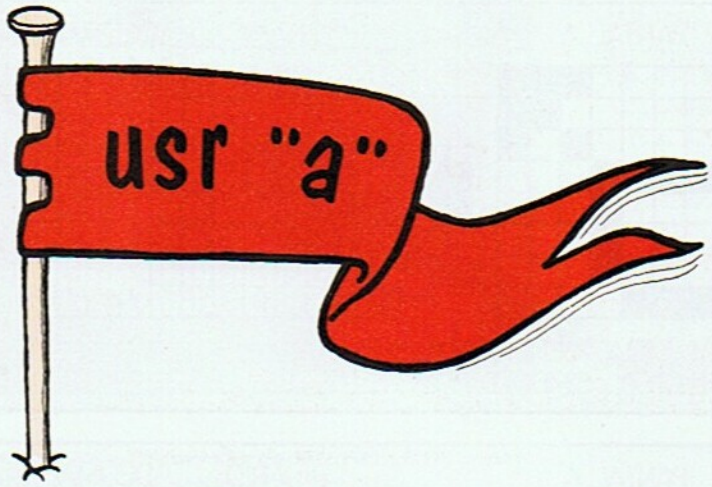
siguiente programa, que asigna a los gráficos de las teclas de la **M** y la **H**, los símbolos representativos de los sexos:

```
 10 REM J.M.LOPEZ MARTINEZ - GENE
RACION DE CARACTERES
 20 FOR I=0 TO 7: READ X: POKE USR
"M"+I,X: NEXT I
 30 FOR I=0 TO 7: READ X: POKE USR
"H"+I,X: NEXT I
 40 PRINT AT 11,12;"M=M, H=H"
 50 DATA 7,3,5,120,136,136,136,112
 60 DATA 56,68,68,68,56,16,56,16
```

En la línea 20 se ejecuta un bucle **FOR NEXT**, que tiene por objeto colocar en la posición **USR "M"** y las 7 siguientes, los valores de las 8 filas del carácter definido, leídas desde la **DATA** de la línea 50, en la variable **X**.
De forma análoga, en la línea 30 se hace lo propio, esta vez a partir de la posición **USR "H"**, y tomando información de la **DATA** de la línea 60. Por último, en la línea 40 se muestra el resultado obtenido.
Los métodos para asignar la forma a un gráfico definido son sin duda variadísimas, y buena prueba de ello la encontraremos en la sección de PROGRAMA, donde la definición de los caracteres de los distintos programas ha sido llevada a cabo de maneras muy diferentes.

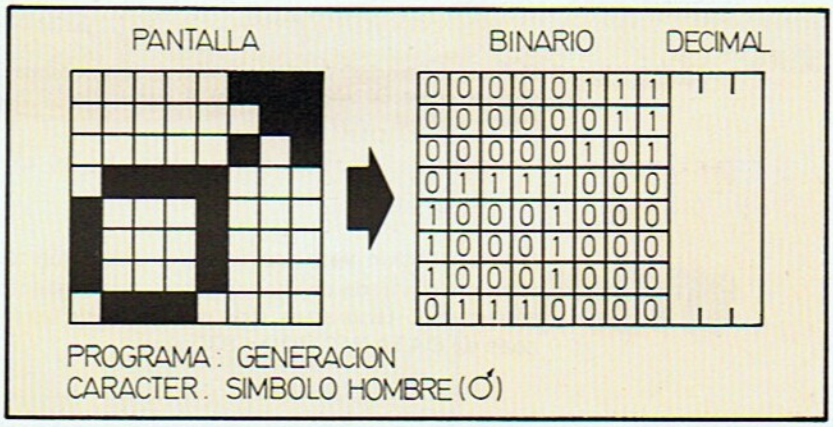
GENERACION DE SPRITES

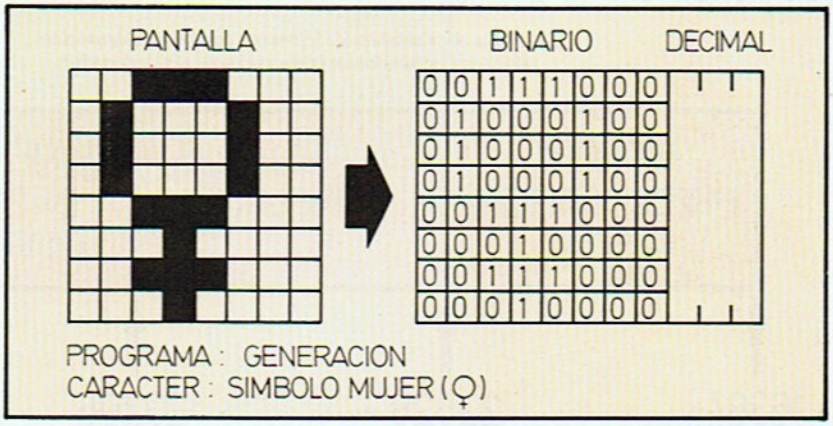
Se conocen como *sprites* los móviles empleados en los programas, cuya forma puede ser redefinida por el usuario. Estos pueden componerse de un sólo carácter pero, generalmente, lo están por un grupo de ellos que se desplazan conjuntamente.
Esto es posible gracias a que las matrices de 8 × 8, que componen los caracteres gráficos, no dejan ninguna separación ni vertical ni horizontal; de este modo, pueden apilarse caracteres definidos unos encima de otros, o de forma lateral, para conseguir un *sprite* de mayor tamaño.
Una muestra de ello, la tenemos en el programa TRUENO AZUL, que ocupa el próximo capítulo de la sección de PROGRAMA. En él, un helicóptero altamente sofisticado es el protagonista de la aventura.

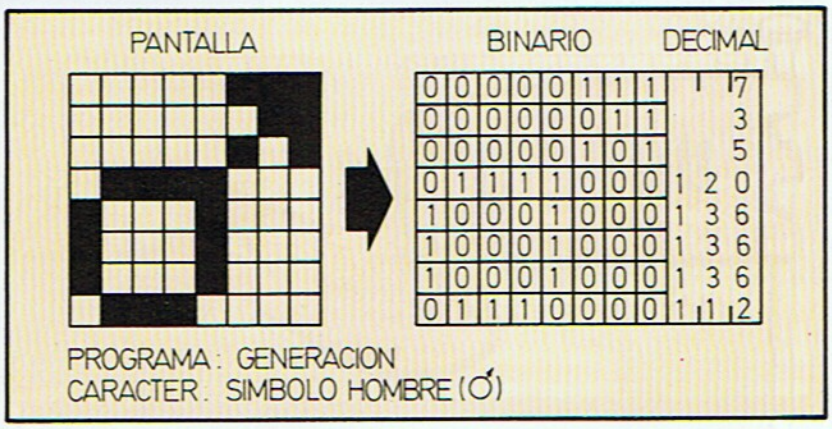


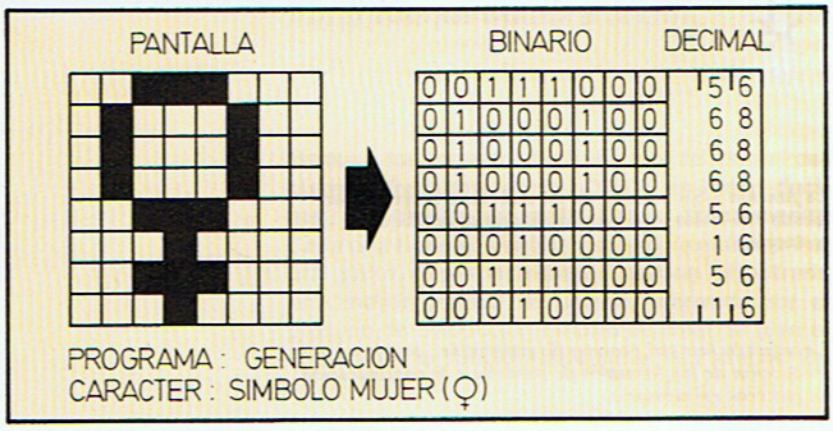
La función USR *seguida de un carácter entrecomillado (desde «a» hasta «u») equivale a una dirección de memoria.*

La segunda fase del proceso de generación, concierne a la traducción de los formatos de pantalla a la configuración de patterns *en memoria.*









Si vamos a introducir los datos en el programa en forma decimal, es preciso un paso anterior de conversión binario-decimal.

A continuación veremos como la alteración de este *sprite* puede ocasionar resultados bastante cómicos: no tendremos más que sustituir las líneas de **DATA** 810, 820 y 830 por:

```
810 DATA 0,35,112,255,255,127,48,3
820 DATA 0,255,129,231,255,255,129,255
830 DATA 0,192,1,253,255,253,1,192
```

Apreciaremos al ejecutar nuevamente el programa como el hilicóptero fruto de la ciencia ficción, se ha convertido en un auténtico Foker superviviente de la Primera Guerra Mundial. ¡Las maravillas de la microinformática!

La tecnología de la generación de un *sprite* suele ser algo más compleja, no tanto por lo que es su inclusión en el programa, que resulta idéntica a la de cualquier gráfico definido, sino por el hecho de que cuando se utilizan varios caracteres solidariamente, hemos de tener mucho cuidado para que coincidan perfectamente, formando un todo único.

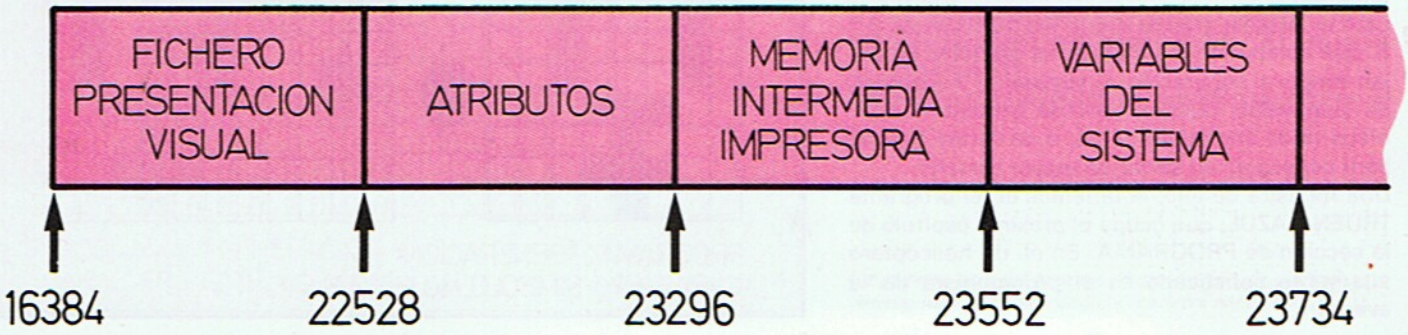
Dado este inconveniente, los programas comunes para generación de gráficos no nos son de gran utilidad en esta aplicación concreta. Así pues, hemos de recurrir, o bien a los programas generadores de pantallas, o bien al método tradicional del lápiz y el papel.

PEEK

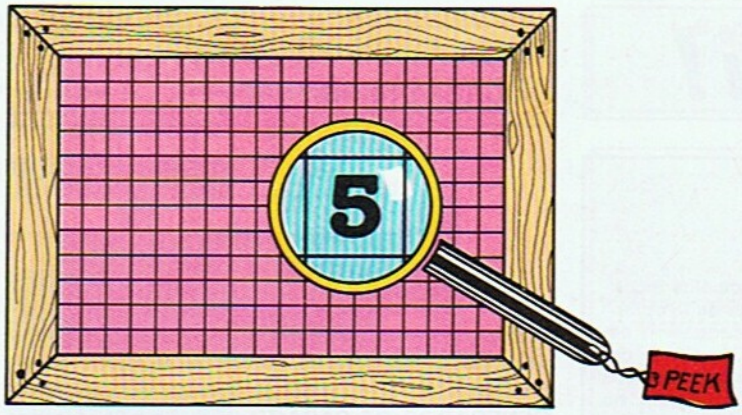
Antes de comenzar la explicación del sistema de reubicación de caracteres, conviene que aprendamos una nueva función BASIC: **PEEK**.

Esta realiza la función inversa a la sentencia **POKE**, es decir, averigua el byte contenido en determinada posición de memoria. Así pues, esta función precisa de un sólo argumento numérico, que deberá estar comprendido entre 0 y 65535. Como en el caso de **POKE**, si el argumento de la función **PEEK** es negativo, recibimos el mensaje

El área de Variables del Sistema contiene información vital para el funcionamiento correcto del ordenador.



La función **PEEK** *investiga el contenido de una dirección de memoria.*



B Integer out of range, y utilizando direcciones no enteras, se efectúa un redondeo hacia el entero más próximo: los argumentos iguales o superiores a 0.5, al inmediatamente superior, y los que no alcanzan dicha fracción, al inferior.
Lógicamente, el resultado de la función **PEEK** será un número entero comprendido entre 0 y 255. Como es natural, puesto que el contenido de la ROM es inalterable, el **PEEK** de cualquier dirección por debajo de 16384 dará siempre un mismo resultado. Por el contrario, las investigaciones dirigidas a la RAM (por encima de 16383) pueden darnos en cada caso diferentes resultados.

La reubicación del generador de caracteres a RAM es un puente tendido a la creación de un número de caracteres definidos, sólo limitado por la cantidad de memoria de que dispongamos.

REUBICACION

rom

ram

¡!

Los principales inconvenientes de la obtención de caracteres especiales mediante caracteres predefinidos son los siguientes: no es posible obtener gran parte de ellos, y si van a ser utilizados frecuentemente en el programa prolongan considerablemente la longitud de éste, además de ser nétamente inferiores en calidad.

*

La principal ventaja del uso de combinaciones de caracteres predefinidos es que no es necesario el proceso de generación.

*

Recurriremos a la reubicación del generador de caracteres cuando sea necesaria la definición de un número de gráficos mayor que 21.

*

Un *sprite* es una unidad gráfica móvil, compuesta generalmente por más de un carácter.

REUBICACION DEL JUEGO DE CARACTERES

Como ya vimos en el capítulo anterior, el juego principal de caracteres, se encuentra definido en una zona de la ROM conocida bajo el nombre de GENERADOR DE CARACTERES. En ella, los *patterns* (forma de los caracteres), se almacenan de la misma manera que lo hacen los gráficos definidos en el RAM; es decir, en conjuntos de ocho bytes, cada uno de los cuales equivale a una línea del carácter, y por tanto, con cada bit representando un punto de la pantalla.

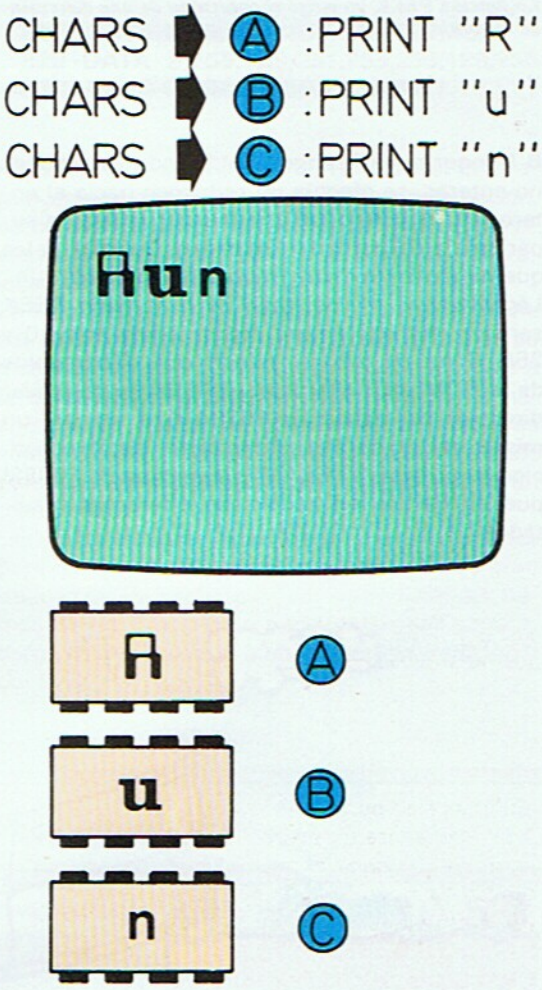
Cada vez que el Spectrum tiene que escribir un carácter, averigua en base a su código de donde debe extraer su forma: si no es representable, es decir, se trata de un carácter de control por debajo del 32, lo sustituye por un signo de interrogación. En caso de que sea un gráfico predefinido, lo construye en una zona especial de la memoria. Tratándose de un gráfico definible por el usuario, localiza su forma actual en la RAM. Pero lo que más nos interesa ahora, es su forma de actuar cuando desea escribir un carácter común, o incluso el mismo signo de interrogación de los no representables.

En este caso, el procedimiento es el siguiente: en primer lugar, averigua cual es la dirección en la cual se encuentra ubicado el generador de caracteres, a continuación cuenta ocho bytes multiplicado por el código del carácter, de forma que se sitúa sobre el *pattern* a representar. Una vez llegado a este punto, todo se limita a leer el punto de la memoria en el cual se encuentra, y los siete posteriores, puesto que inicialmente está en la primera línea del carácter, para ir volcando sus contenidos en la pantalla, una línea debajo de otra.

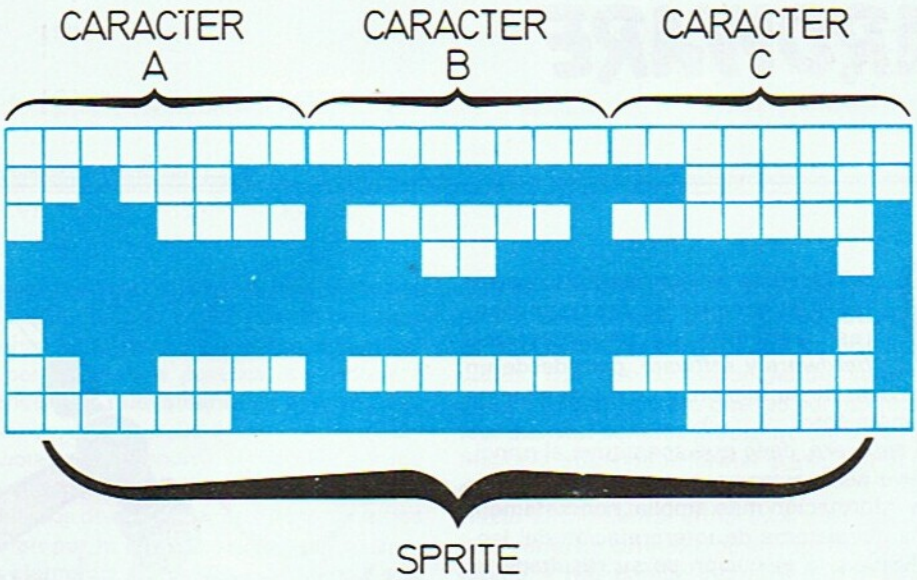
En todo este proceso, debe haber dos cosas que nos llamen la atención: por una parte, el hecho de que sea necesario averiguar en que lugar se encuentra ubicado el generador de caracteres, dado que si la ROM no se altera, siempre estará en la misma posición. Por otra parte, hemos dicho que contamos ocho multiplicando por el código del carácter a localizar, y si consideramos que el generador de caracteres comienza con el *pattern* del espacio (los 32 primeros caracteres son los de control), deberíamos antes de realizar la multiplicación, restar ocho al código de carácter a buscar. He aquí las explicaciones a estos dos hechos.

La programación contenida en la ROM, necesita de una zona de RAM en la cual poder volcar información vital para el correcto funcionamiento del ordenador, como por ejemplo, en que color debe escribir a continuación, si la cantidad de memoria dispinible son 16 K o 48 K, en que modo de cursor nos encontramos, y muchísimas otras. A tal fin, existe un área reservada de la memoria RAM, situada detrás de la pantalla, y antes del lugar donde se almacenan nuestros programas BASIC, que se denomina AREA DE VARIABLES DEL SISTEMA.

Entre las muchas Variables del Sistema, se encuentra una que le indica al ordenador donde se localiza el generador de caracteres, denominada simbólicamente CHARS; decimos «simbólicamente» porque estas variables y sus nombres no tienen nada que ver con nuestras propias variables BASIC, y le otorgamos este nombre (CHARS)



Mediante el manejo adecuado del puntero CHARS podemos simultanear en pantalla caracteres pertenecientes a varios generadores.

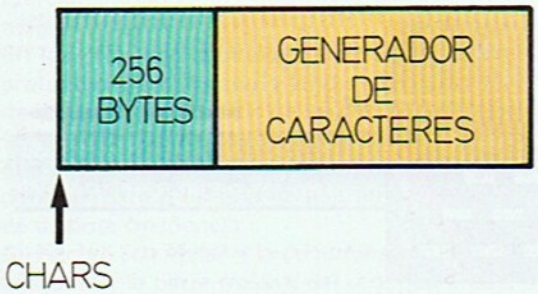


Un sprite *es una unidad gráfica móvil, compuesta generalmente por más de un carácter.*

simplemente porque es el que se le da en el manual del aparato. Al encender el ordenador, y siempre que no alteremos el valor de esta Variable del Sistema, esta señala al generador de caracteres en ROM. No obstante, su situación en RAM, nos da la posibilidad de alterar el valor de CHARS para que señale a otro punto.

Este otro punto, lógicamente, es una zona de la RAM, donde nosotros podemos haber depositado los *patterns* de todo un juego de caracteres; a esto se le denomina REUBICACION DEL GENERADOR DE CARACTERES. Naturalmente, si CHARS apunta a un lugar no apunta a otro, es decir, no podemos disponer simultáneamente del juego de caracteres de ROM y del nuestro propio de RAM,

La Variable del Sistema CHARS siempre señala 256 bytes antes del comienzo del generador de caracteres.



aunque dentro de unas líneas tendremos la oportunidad de ver un sistema para simular o. Pero antes queda por resolver una cuestión pendiente. El hecho de que no restemos 32 al código de carácter antes de iniciar la búsqueda del carácter (multiplicar por ocho su código) se debe a que para evitarnos este problema, CHARS no apunta exactamente al comienzo del generador de caracteres, sino concretamente 256 bytes antes: 8 bytes por 32 caracteres suponen 256 bytes, a sabiendas de que el sistema de 256 bytes (32 caracteres) correspondientes a los códigos de control, y que ya hemos comentado con anterioridad.

En cuanto a la forma de simultanear, aunque sólo sea de manera aparente, varios juegos de caracteres (por ejemplo el de la ROM y uno o varios en RAM), se basa en el hecho de que una vez escrita la información en la pantalla, ésta no se altera. Así pues, nosotros podemos escribir en el juego de caracteres que queramos, sólo con modificar previamente CHARS para que apunte al generador correspondiente en cada caso.

Para terminar veámos el sistema de alteración de CHARS. Esta Variable del Sistema se encuentra en las direcciones 23606 y 23607. Por tanto, siendo X la nueva situación deseada para el generador, en primer lugar le restamos 256 (**LET X=X-256**) y luego tan sólo es necesario efectuar un **POKE** en 23607 de la parte entera de X partido por 256 (**POKE 23607,INT (X/256)**, y a continuación, de su resto en 23606 (**POKE 23606,X-256*PEEK 23607**).

En general, podemos utilizar la siguiente línea BASIC, siempre y cuando recordemos asignar a **X** la dirección de comienzo del generador requerido antes de ejecutarla (**LET X=<dirección del generador>): LET X=X-256: POKE 23607, INT(X/256): POKE 23606, X-256* PEEK 23607.**

¡!

Gracias a la generación de caracteres se pueden obtener resultados de gran calidad.

*

La Variable del Sistema que indica la situación del generador de caracteres se denomina CHARS.

*

CHARS señala siempre 256 bytes antes de la dirección de comienzo del generador de caracteres.

*

Un *pattern* es el formato en bytes que un carácter tiene en la memoria.

*

El área de Variables del Sistema contiene información vital para el correcto funcionamiento del ordenador.

HARDWARE

HASTA ahora hemos tenido la oportunidad de estudiar los dos componentes esenciales de un ordenador: *hardware* y *software*, pero desde un punto de vista de su misión, y no de su sistema de funcionamiento.

Sobre el *software*, dado que constituye el principal objetivo de nuestro aprendizaje, hemos recibido una información más amplia: concretamente, acerca del sistema de interpretación del lenguaje BASIC, y la ejecución de su resultado: el código máquina.

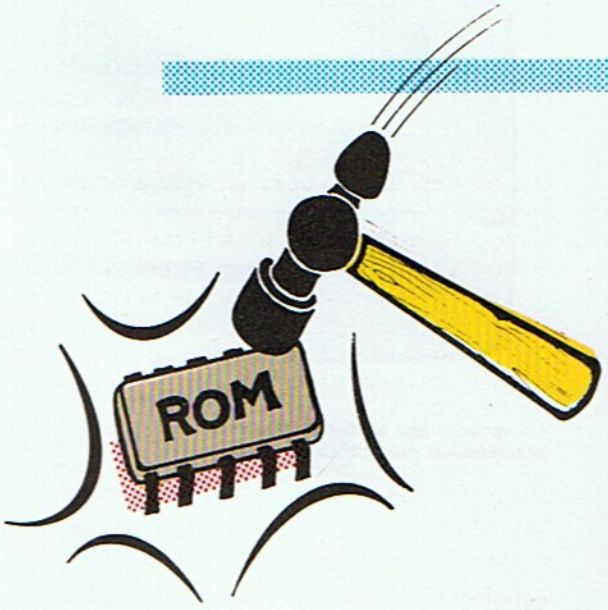
Ahora bien, el *hardware* fue también estudiado desde la óptica de su misión, descuidando lo referente a su organización interna para conseguir un funcionamiento tan sumamente eficaz. De esto nos ocuparemos en el presente capítulo.

Como ya sabemos, el *hardware* es el conjunto de elementos físicos que integran nuestro ordenador. Todos ellos tienen por fin último el tratamiento lógico de una información. Es aquí donde se haya la propia esencia del sistema: en la información.

Lo que nosotros entendemos por información, no se parece prácticamente en nada a lo que el ordenador entiende como tal. Realmente, la información según la vamos a tratar ahora, desde el punto de vista de nuestro Spectrum, no es más que un simple estado de la corriente eléctrica. Dado que el *hardware* no nos sirve para nada sin la información memorizada en su *firmware*, veamos primeramente como funcionan las memorias. Reduciendo enormemente su complejidad técnica, la memoria no es más que un conjunto de circuitos interrelacionados, basados en un elemento capaz de retener un cierto nivel de corriente eléctrica, que nosotros asociamos a un estado lógico (1 ó 0).

Cada uno de estos elementos es lo que para nosotros constituye un bit, y la relación de ocho de ellos, un byte. Por último, la cantidad total de ellos que integran estos circuitos, ya sea en uno o en varios *chips*, conforman el conjunto de la memoria del ordenador.

Realmente, existe una distinción entre las memorias RAM y ROM, dado que las primeras pueden alterar el nivel de corriente de estos elementos básicos, mientras que el contenido de una memoria ROM es físicamente inalterable. De ahí que, ni aún intencionadamente, consigamos dañar permanentemente nuestro aparato utilizando la programación.



El contenido de una memoria ROM es inalterable por medio de la programación. Mucho nos tememos que para su destrucción haya que recurrir al martillo.

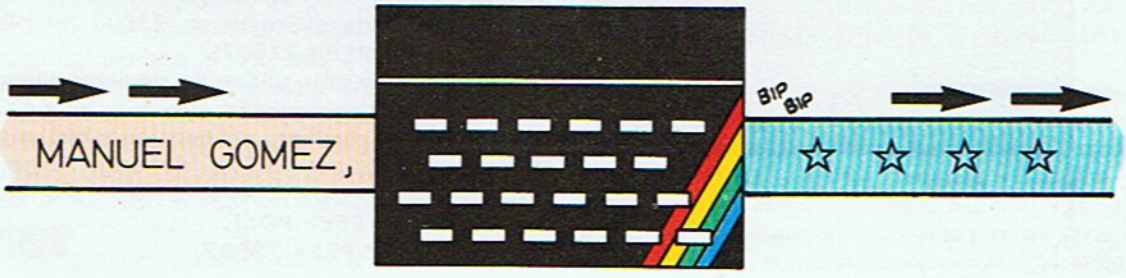
¡!

Las puertas lógicas fundamentales realizan las funciones lógicas booleanas: AND, OR, XOR y NOT.

*

Existen dos puertas utilizadas bastante frecuentemente: NAND y NOR. La puerta NAND equivale a una puerta AND cuya salida es negada. La puerta NOR, también conocida como puerta NI, equivale a una puerta OR cuya salida es negada.

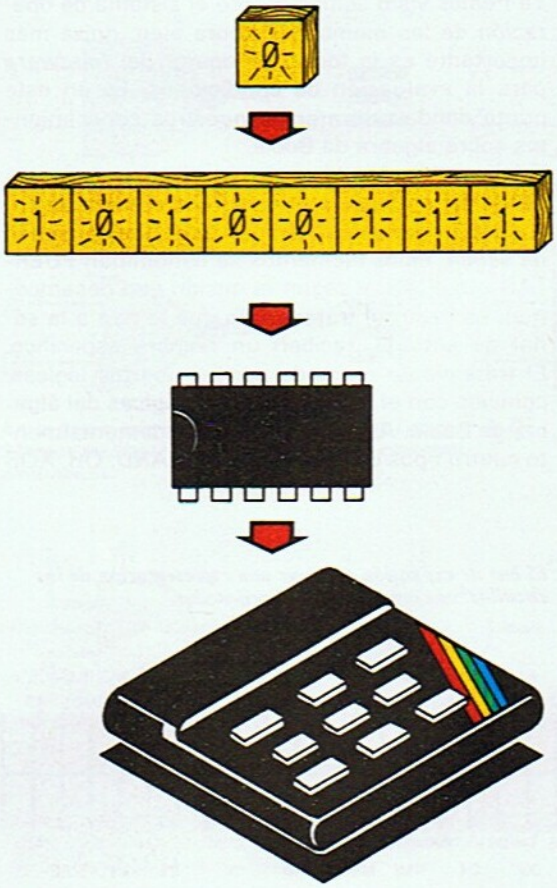
El hardware *trata la información como simples impulsos eléctricos.*



MANUEL GOMEZ,

Ni tan siquiera desde el código máquina podemos alterar el contenido de la ROM, puesto que esto es físicamente imposible, y por tanto, cualquier problema, por grave que parezca, se reduce a apagar y encender nuevamente el aparato. Así que ya sabemos; si queremos destruir nuestro ordenador, habremos de recurrir al martillo, porque nunca lo conseguiremos con la programación.
Estos dos estados de corriente, que denominamos 0 y 1, circulan por la circuitería de nuestro aparato de un elemento a otro: de la memoria a la C.P.U. y viceversa, entre los *ports* y la C.P.U. através de la U.L.A., etc...
En definitiva, las pistas del circuito impreso sobre el cual se encuentran los componentes del ordenador, son como las calles de una ciudad extraordinariamente transitada, recorridas por centenares de impulsos eléctricos, con los cuales la inteligencia humana ha sido capaz de codificar información.
Algunas de estas señales eléctricas, tienen el sentido de información, y circulan através de los buses, tanto de datos como direcciones, en los cuales se expresan, respectivamente, el dato a transportar, y el lugar del ordenador al cual va destinado. De nuevo, pese a esta importante misión que le atribuimos a los buses, estos no son más que un conjunto de pistas por las cuales circula la corriente eléctrica.
Por otros circuitos, llamémosles menos especializados, o de segundo orden, circulan señales con otros sentidos, como es propiamente la corriente eléctrica de alimentación, que le da a los componentes la energía necesaria para su funcionamiento.
Un ejemplo muy claro de este hecho, es la transmisión de datos a la grabadora de casetes. En ella vemos claramente, «oímos» este aspecto. Así por ejemplo, esos unos y ceros, que dentro del ordenador adquirirán el sentido de un apasionante juego, o un complicado programa de gestión comercial, no son más que unos estridentes sonidos.
En los próximos capítulos estudiaremos más en profundidad este hecho, y tendremos oportunidad de ver la representación en un osciloscopio de estas señales. De momento, nos bastará con saber, que cuando se emite o se recibe desde la grabadora, un cero o un uno no son más que señales de distinta frecuencia.
De hecho, con vistas a la comunicación con otras máquinas, la parte trasera del ordenador deja al descubierto una parte de la tarjeta de circuito impreso, denominada BUS DE EXPANSION. En el se encuentran representados los principales circuitos, con sus correspondientes señales, para que cualquier aparato que las comprenda, pueda leer o enviar señales (información) a través de los mismos, pasando a formar parte, como periférico, del sistema ordenador.
Hasta el momento, el *hardware* se nos ha revelado como un conjunto de elementos bastante estúpido, y nos parece increible que el simple hecho de enviar corriente en dos estados a través de unos circuitos, que no dejan de ser meros cables, pueda llegar a dar resultados que en ocasiones pueden parecer inteligentes.

La asociación de los componentes básicos de la memoria, capaces de memorizar un estado de corriente (bit), conducen a la construcción de un conjunto de varios Kbytes.



¡!

En los esquemas lógicos, las puertas NAND se representan como puertas AND con un círculo antes de su salida, y las NOR con el círculo antes de la salida de una puerta OR.

*

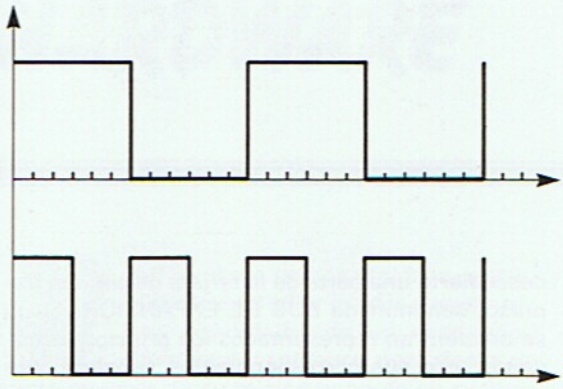
La lógica triestado admite tres estados diferentes de corriente: dos de ellos se asignan a los niveles lógicos (1 y 0), y el tercero a la ausencia de éstos.

*

La cadencia de operación del ordenador viene marcada por el reloj.

¡!

Existe una imposibilidad física para alterar el contenido de una memoria ROM mediante la programación; por tanto, no nos debe importar experimentar con los **POKEs**, ni desde el BASIC, ni desde el propio código máquina.



Las informaciones almacenadas en un casete no son más que ondas de diferente frecuencia.

*

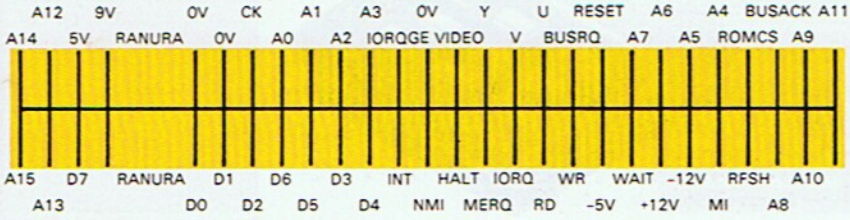
Las puertas lógicas son elementos del *hardware* capaces de dar resultados distintos en base a sus entradas.

La «inteligencia» del ordenador reside fundamentalmente en dos puntos: su capacidad de memorizar estados de corriente (información), y su capacidad para alterar estos estados (información) de acuerdo con determinadas condiciones.
Ya hemos visto someramente el sistema de operación de las memorias. Ahora bien, quizá más importante es la forma de operar del *hardware* para la evaluación de condiciones. Es en este punto donde utilizaremos nuestros conocimientos sobre álgebra de Boole.
Existen unos elementos del *hardware* que son capaces de, en base a un estado de entrada de corriente (información), dar un determinado estado de salida. Estos elementos se denominan PUERTAS LOGICAS, y según la misión que desempeñen, es decir, el tratamiento que le den a la señal de entrada, reciben un nombre específico. El tratamiento otorgado por las puertas lógicas, coincide con el de las funciones lógicas del álgebra de Boole. Así pues, existen fundamentalmente cuatro tipos de puertas lógicas: AND, OR, XOR y NOT (conocida generalmente como puerta INVERSORA).
En los esquemas de circuitería, cada una de estas puertas se haya representada mediante un símbolo, que podemos observar en las figuras adjuntas.
Estas puertas, son capaces de tratar un bit de información por cada entrada, generando una señal (bit) de salida. Lógicamente, podemos combinar ocho puertas de un mismo tipo para efectuar una función lógica sobre un byte completo.
Como vimos anteriormente, mediante el álgebra de Boole, podemos tratar matemáticamente problemas que aparentemente no tienen nada que ver con esta ciencia; como por ejemplo, si nuestro coche puede o no atravesar un paso a nivel. Gracias a ello, el ordenador puede gestionar la información de una manera aparentemente «inteligente», basando sus decisiones en el resultado de salida de las puertas lógicas.
Además de estas cuatro puertas lógicas fundamentales, se suelen emplear dos más: NAND y NOR. La primera equivale a la combinación de una puerta AND con un NOT a su salida, y la segunda, a una puerta OR a cuya salida se ha aplicado también una inversora.
En el último capítulo, estudiamos un importante teorema del álgebra de Boole: el Teorema de De Morgan. Su aplicación principal se da en la construcción de circuitos. Según él, podemos construir una función AND negando las entradas aplicadas a una puerta NOR, e igualmente, es posible construir una función OR mediante la negación de las entrada de una puerta NAND. Esto debe servirnos a modo puramente anecdótico, a no ser que nos dediquemos al diseño de algún circuito para conectar al Spectrum.

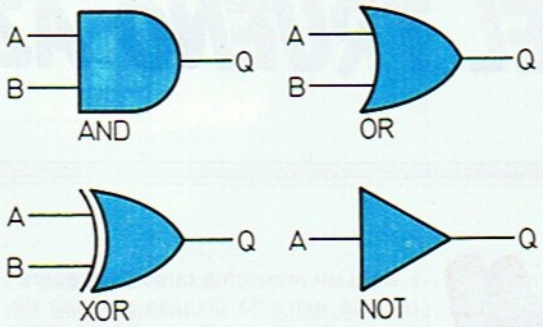
SINCRONISMO

Otro punto importante en el comportamiento del *hardware*, es el sincronismo de sus acciones. No debemos pensar que la nube de señales eléctricas (informaciones) que fluyen por las pistas, deambulan de una manera anárquica e incontrolada. Existe por el contrario un sistema de regulación, que sincroniza todos los tratamientos de información.
Este sistema que ya pudimos estudiar con anterioridad se denomina SINCRONISMO, y el núcleo fundamental del mismo, se encuentra en un oscilador (cristal de cuarzo), conocido bajo el nombre de RELOJ.

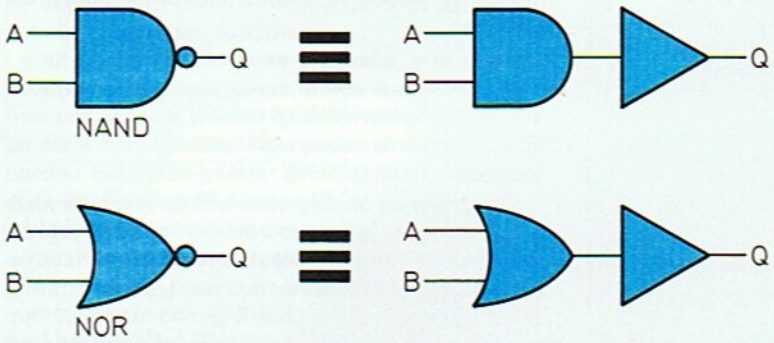
El bus de expansión contiene una representación de los circuitos más importantes del ordenador.



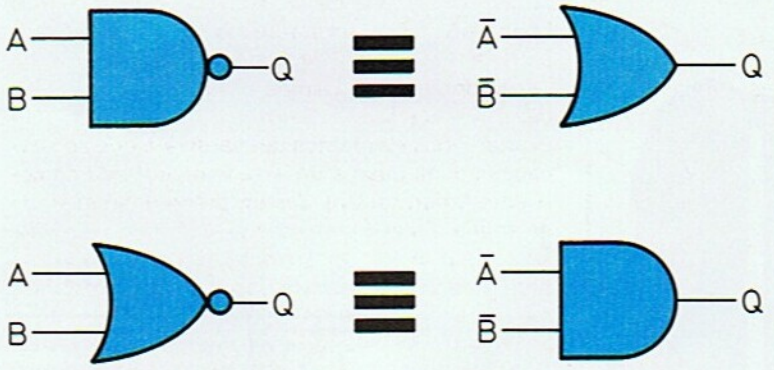
Este reloj envía una señal a intervalos iguales de tiempo, y esta señal es aguardada por el *hardware* para realizar todas las operaciones.
Supongamos que nos encontramos ante una puerta AND, a la cual llegan dos informaciones; sin embargo, el resultado no aparece a su salida ¿qué es lo que ocurre? si ya tiene los dos operandos, ¿a qué espera la puerta AND para procesar la información? Simplemente, a la llegada del impulso del reloj (señal de sincronismo), que le indique que puede trabajar.
En cierto modo, el reloj se puede comparar con los golpes de bombo, que en las antiguas galeras de esclavos marcaban la cadencia de las paladas. Por cada golpe, se remaba una palada; cuanto más seguidos eran los golpes, más rápido se remaba.
Análogamente, la velocidad de ejecución de un ordenador depende directamente de la cadencia que marque su reloj. En el caso concreto del Spectrum es de aproximadamente 3.5 Mhz (Megahercios), lo cual supone que se emite una señal de sincronismo unos tres millones y medio de veces por segundo.



Las puertas lógicas conciden con las funciones lógicas del álgebra de Boole.



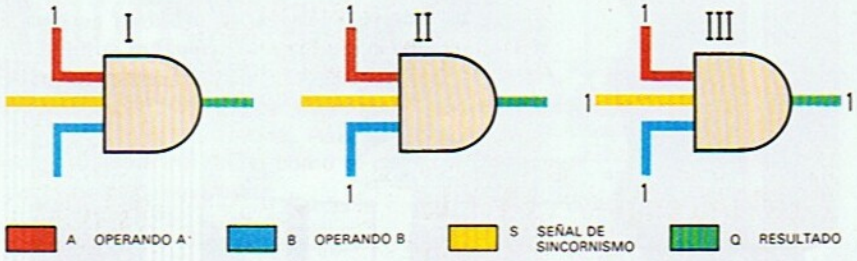
Una puerta NAND equivale a la combinación de una puerta AND y una puerta NOT. Análogamente, una puerta NOR equivale a la combinación de una puerta OR y una puerta NOT.



En la figura aparece la representación del Teorema de De Morgan mediante símbolos lógicos para esquemas electrónicos.

LOGICA TRIESTADO

Para finalizar diremos que, hasta ahora, hemos supuesto que existen dos estados de corriente, cuyo significado lógico es uno o cero; sin embargo, esto es cierto sólo en parte. Supongamos que somos un periférico reclamándole una información al Spectrum. Acudimos a su bus de expansión y comprobamos el estado lógico de determinado circuito.
Si el nivel es uno, tendremos claro que esa es la información que nos desea transmitir. Ahora bien, si es un cero, ¿cómo saber si ese es el bit que quería transmitir el ordenador? pudiera ser que le hubiéramos sorprendido en un momento de «distracción», y simplemente no deseara transmitirnos ningún dato.
De esta circunstancia nace la necesidad de un tercer estado de corriente, esta vez sólo al nivel del *hardware*, de forma que además de los dos estados de corriente asociados a los bits (1 ó 8), exista un tercero que indique la falta de nivel lógico.
Este sistema, en el cual se utilizan tres estados: dos para representar los lógicos (1 ó 8) y el tercero para indicar la falta de estado lógico, se le denomina LOGICA TRIESTADO, y es la empleada por el *hardware* de nuestro ordenador.



Debido al sincronismo, una puerta lógica no sólo debe esperar la llegada de sus operandos, sino también la señal que le permite ejecutar su función.

EL TRUENO AZUL

HE aquí un programa cuyo protagonista está extraído de una película de ciencia ficción: el Trueno Azul *(Blue Thunder)*. Este es el avión más veloz, sofisticado, destructivo, poderoso, y temido del mundo, y nosotros somos los encargados de pilotarlo.

Nuestra misión es destruir lo más rápidamente posible las edificaciones que el enemigo ha construido en nuestro territorio, con el objetivo de utilizar el terreno como pista de aterrizaje para el aparato, y efectuar desde allí nuevas incursiones. A tal fin, el Trueno Azul ha sido equipado con bombas antimateria. Estas peligrosas armas son

capaces de destruir toda una columna de edificios, dejándola reducida a cenizas con solo tocarla. Debido a la gran velocidad que posee nuestro avión, el selector automático de disparo sólo nos permite efectuar tres lanzamientos por pasada, por lo cual no debemos desaprovechar ninguna bomba.

La casi totalidad de los microordenadores, cuentan en su biblioteca de programas con un juego similar al que hoy presentamos.
Pilotamos una nave que vuela cada vez más bajo, y debe destruir todos los bloques de edificios que se encuentran debajo de ella, para poder aterrizar sin chocar con ninguno.
La altura de las casas es aleatoria, y el número de bombas de que disponemos está limitado a tres por pasada, puesto que de no ser así, el juego sería muy sencillo. Para poder efectuar el bombardeo debemos pulsar **SPACE** (barra espaciadora en el Plus). Por cada bloque de edificio destruido, el Spectrum nos concede un punto. Si el arrasamiento ha sido total, los puntos obtenidos (puntos totales) son conservados y sumados a los que vayamos consiguiendo en la nueva pantalla.
Para hacer más complicado el juego, el avión que pilotamos desciende de dos en dos filas por cada pasada, debido a lo cual, no sólo debemos tener buena puntería para destruir los edificios, sino seleccionar cuidadosamente los blancos de nuestra acción; de no ser así, lo más seguro es que no duremos mucho tiempo sin toparnos con un edificio.
El programa posee además un marcador de records, en el cual queda almacenada la mayor puntuación lograda hasta el momento.

EL PROGRAMA

Como es habitual, la introducción del programa no plantea más problema que la presencia de los caracteres subrayados. Estos sutituyen a los gráficos definidos cuya tecla tiene por carácter más significativo el subrayado. Así por ejemplo, la A subrayada se introduce como el carácter gráfico de la tecla A.

El disparador de nuestro «Trueno Azul» es la tecla de **SPACE.**

¡!

El *Blue Thunder*, antes de posarse en el suelo, necesita destruir todos los edificios que aparecen en la pantalla.

*

El lanzamiento de las bombas antimateria se consigue pulsando **SPACE**.

*

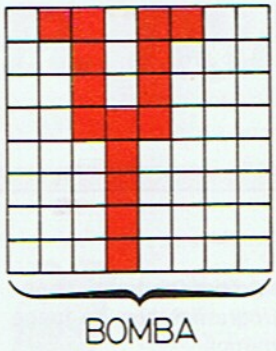
En el listado del programa, los caracteres subrayados corresponden a los gráficos definidos de usuario.

*

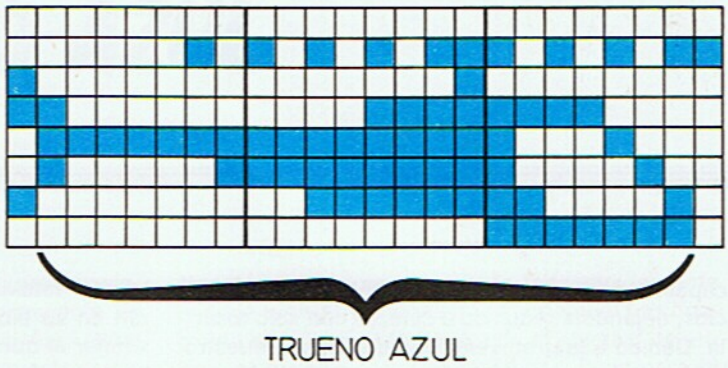
A la hora de grabar el programa, utilizaremos el siguiente comando: **SAVE "B.THUNDER"**. Si optásemos por la opción de autoejecución, tecleríamos la secuencia **SAVE "B.THUNDER" LINE 10.**

*

Para hacer más difícil el juego, nuestro avión desciende dos líneas por pasada.



En el gráfico definido de la C se encuentra la bomba antimateria.



El «Trueno Azul» viene definido por los tres primeros gráficos de usuario.

Recordemos que para introducir un carácter gráfico, hemos de pasar el cursor a modo G; o bien pulsando la tecla **GRAPH** en el Plus, o bien pulsando simultáneamente las teclas **CAPS SHIFT** y **9**, sistema éste válido tanto para el ZX Spectrum, como para el Plus. Una vez introducido el gráfico o gráficos precisos, debemos desconectar el modo **G**, para continuar con la introducción del resto del listado. Ello se consigue, ya sea repitiendo los pasos dados para la conexión, como pulsando simplemente la tecla **9** (válida para los dos modelos).

La estructura del programa es francamente clara, y ayudados por los **REM** que se incluyen en el mismo, se comprensión no estriba inconveniente alguno. Ahora bien, es destacable el sistema que hemos seguido para la generación de edificios, dado que gracias a un único grafico definido, podemos crear «rascacielos» de la altura que deseemos.

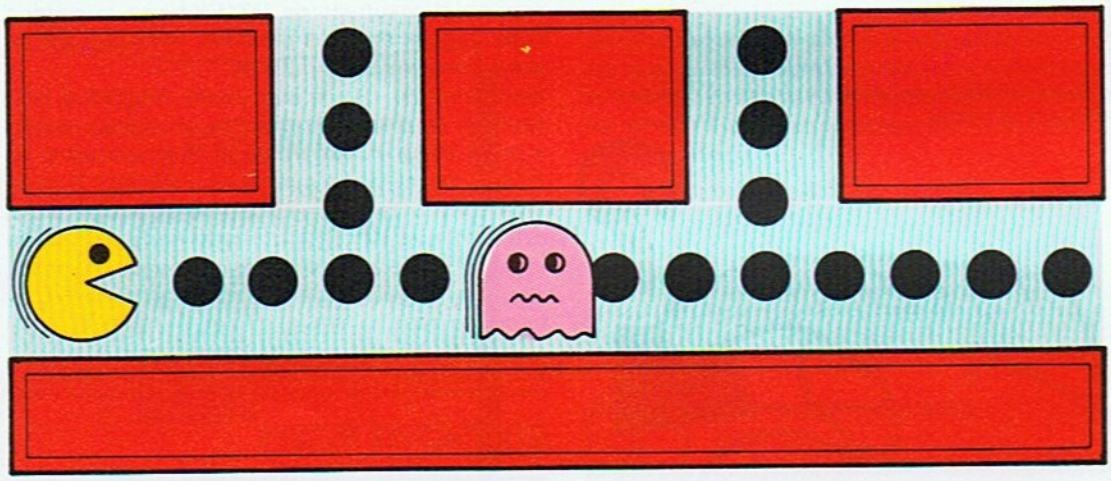
A la hora de la grabación del programa, sigamos el sistema habitual: **SAVE "B.THUNDER"**. Si deseamos que el programa se ejecute al finalizar su carga, debemos grabarlo mediante **SAVE "B.THUNDER" LINE 10**. ¡Buen aterrizaje!

```
  10 REM ************************
  20 REM * J.M.MAYORAL SERRANO *
  30 REM ************************
  40 REM * BLUE THUNDER   1985 *
  50 REM ************************
  60 REM   INIC. VARIABLES
  70 LET REC=0
  80 LET REC1=0
  90 LET SW1=0
 100 LET PT=0
 110 LET PUN=0
 120 REM   COMIENZO
 130 GO SUB 790
 140 GO SUB 260
 150 GO TO 700
 160 LET X=20-FILA
 170 FOR Q=0 TO X
 180 PRINT  INK 5;AT FILA,COL;"ABC"
 190 PRINT  INK 6;AT FILA+Q,COL;"E"
 200 BEEP .001,50
 210 PRINT AT FILA+Q,COL;" "
 220 PRINT  PAPER 4;AT 20,8;" "
 230 IF ATTR (FILA+Q+1,COL)=58 THEN  LET PUN=PUN+1: P
RINT "PUNTOS ";PUN
 240 NEXT Q
 250 RETURN
 260 REM   DIBUJO EDIFICIOS
 270 LET F1=20
 280 FOR W=1 TO 31
 290 PRINT  PAPER 4;AT F1,W;" "
 300 NEXT W
 310 PRINT  PAPER 2;AT 21,18;"  RECORD=      ";AT 21,2
8;REC
 320 REM
 330 REM
 340 FOR Z=4 TO 28
 350 FOR J=INT (RND*7)+12 TO 19
 360 PRINT  PAPER 0;AT J,Z;" "
 370 PRINT ; INK 2; PAPER 7;AT J,Z;"D"
 380 NEXT J
 390 NEXT Z
 400 LET CERO=0: LET CAB=0
 410 REM  MOVIMIENTO HELICOPTERO
 420 REM
 430 FOR A=0 TO 20 STEP 2
 440 LET SW=1
 450 FOR B=0 TO 28
 460 LET FILA=A: LET COL=B
 470 PRINT  INK 5;;AT FILA,COL;"ABC"
 480 BEEP .001,40:
 490 REM COPMR.IMPACT & VICT.TOT
 500 IF ATTR (FILA+CAB,COL+CERO+4)=58 THEN  LET CERO=
CERO-2: LET FILA=FILA: IF FILA=FILA THEN  GO TO 580
 510 IF FILA=20 AND COL=28 THEN  LET SW1=1: PRINT  FL
ASH 1;AT 10,10;" BUEN PILOTO ": LET PT=PT+PUN: PRINT
 INK 9;AT 20,0;"PUNTOS TOTALES ";PT: GO SUB 990: GO T
O 710
 520 IF SW>=4 THEN  GO TO 540
 530 IF INKEY$=" " THEN  LET SW=SW+1: GO SUB 160
 540 LET FILA=FILA: LET COL=COL
 550 NEXT B
 560 PRINT  PAPER 1;AT FILA,29;"   "
 570 NEXT A
 580 FOR D=FILA TO 20
 590 REM   CHOQUE
 600 REM
 610 BEEP .001,50
 620 PRINT  INK 2;AT FILA,COL+2;"AB": FLASH 1;"D"
 630 PRINT AT FILA,COL+1;" "
 640 NEXT D
 650 REM  TRASVASE DE PUNTOS
 660 LET PT=PUN+PT
 670 RETURN
 680 REM.  FIN DE JUEGO
 690 REM
 700 PRINT AT 5,8;"PUNTUACION= ";PT
 710 IF SW1 THEN  GO TO 930
 720 GO SUB 990
 730 PRINT  PAPER 2; FLASH 1;AT 21,28;REC
 740 PRINT AT 7,7; PAPER 6;" OTRA PARTIDA ? "
 750 LET K$=INKEY$
 760 IF K$="S" OR K$="s" THEN  BEEP .2,40: GO TO 90
 770 IF K$="N" OR K$="n" THEN  GO TO 10000
 780 GO TO 750
 790 REM   GRAFICOS USUARIO
 800 RESTORE 810
 810 DATA 0,2,128,192,127,65,128,0
 820 DATA 0,171,1,15,255,255,63,0
 830 DATA 0,171,0,240,136,132,194,254
 840 DATA 255,255,137,137,255,137,137,137
 850 DATA 108,40,40,56,16,16,16,16
 860 FOR K=USR "A" TO USR "E"+7
 870 READ A
 880 POKE K,A
 890 NEXT K
 900 PAPER 1: BORDER 1: INK 9
 910 CLS
 920 RETURN
 930 FOR N=1 TO 100
 940 BEEP .01,20: BEEP .01,40
 950 NEXT N
 960 LET SW1=0
 970 GO TO 110
 980 REM   COMPROBACION RECORD
 990 IF PT>REC THEN  LET REC=PT: GO TO 1010
1000 RETURN
1010 PRINT AT 0,4; FLASH 1; PAPER 7;"ESTABLECIO UN NU
EVO RECORD";AT 1,10;"EN "; INK 7; PAPER 9;REC; PAPER
7; INK 9;" PUNTOS"
1020 RETURN
```

EN MOVIMIENTO

COMO ya vimos en el capítulo anterior, se conocen como *sprites* los móviles empleados en los programas, generalmente presentes en juegos de acción, pero también en educativos o programas de cualquier otro tipo. Los *sprites* son unidades gráficas que pueden componerse de un sólo carácter, pero usualmente lo están por un grupo de ellos que se desplazan conjuntamente. Esto es posible gracias a que las matrices de 8 × 8, que componen los caracteres del Spectrum, en base a los cuales se construyen los móviles, no dejan ninguna separación ni vertical ni horizontal entre ellos; de este modo, pueden apilarse caracteres definidos unos encima de otros, o de forma lateral, para conseguir un *sprite* de mayor tamaño.

En este mismo capítulo, encontraremos un juego de acción, en el cual será premiada nuestra puntería a la hora de abatir aviones enemigos, que sobrevuelan negligentemente nuestras baterías antiaéreas.



Los sprites *son unidades gráficas dotadas de movimiento, las cuales pueden estar compuestas por uno o más caracteres.*

El programa se basa en los conocimientos ya adquiridos de como generar caracteres del usuario, producir efectos de color, y otros muchos; pero entre ellos se hace uso de una sentencia nueva: **INKEY$**. La importancia de esta nueva función, merece que le dediquemos una explicación previa, antes de entrar al comentario general de su funcionamiento en el programa.

INKEY$

Básicamente, la función **INKEY$** nos permite averiguar el último carácter pulsado en el teclado, y utilizar dicho valor de cualquier forma que deseemos.

Así pues, esta función nos devuelve un sólo carácter, para que pueda ser almacenado en una variable, impreso en la pantalla, etc... dado que en su forma de función, la flexibilidad para el manejo del resultado es enorme, y depende directamente de la palabra clave que, obligatoriamente, debe anteponerse a la propia función.

Hasta ahora, y puesto que ya conocemos la existencia de **INPUT**, esta función no nos parece muy importante, sino más bien un caso excesivamente restringido de dicha sentencia. No obstante, aunque ambas sentencias estén destinadas a la toma de datos, la diferencia entre ellas es sustancial: **INKEY$** no detiene la ejecución del programa.

Si la toma de datos del programa hubiera sido

¡!

Dado que las matrices de 8 × 8 que configuran los caracteres no presentan ninguna separación al ser impresas contiguamente en la pantalla, podemos generar unidades gráficas continuas en base a caracteres simples.

*

INKEY$, como cualquier otra función, debe ir precedida de una palabra clave, que indique la forma de aplicar el resultado.

*

INKEY$ permite averiguar el último carácter pulsado en el teclado.

¡!

INKEY$ no considera caracacteres significativos, de manera independiente, ninguno de los dos **SHIFT: CAPS** o **SYMBOL**.

*

INKEY$ consulta el teclado en el momento de su ejecución, pero no detiene el programa.

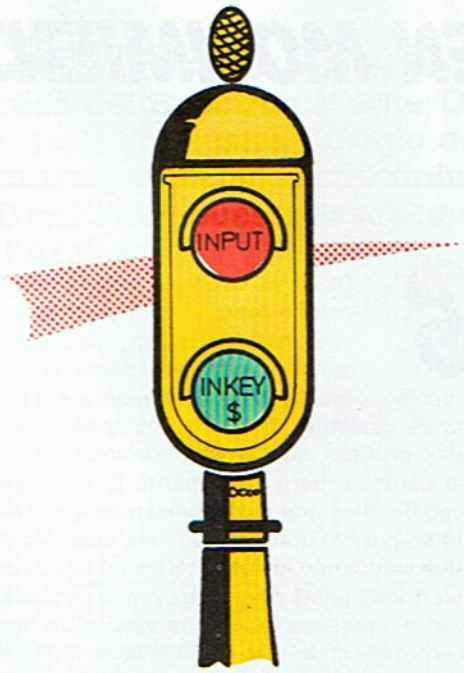
realizada por **INPUT**, el objetivo de nuestros disparos se iría deteniendo cada vez que avanzara un carácter en la pantalla y, de esta manera, sería evidentemente sencillo lanzar un proyectil en el momento adecuado.

Así pues, siempre que deseemos que una acción continúe, ya sea el movimiento de un *sprite* o cualquier otra, y que al mismo tiempo se efectúe una toma de datos, es imprescindible el uso de la función **INKEY$**.

Lógicamente, dos cosas no pueden suceder al mismo tiempo, y si nos encontramos desplazando el móvil, no estaremos realizando la toma de datos; sin embargo, las cosas se sucederán a tal velocidad en el programa, que tendremos una sensación de simultaneidad.

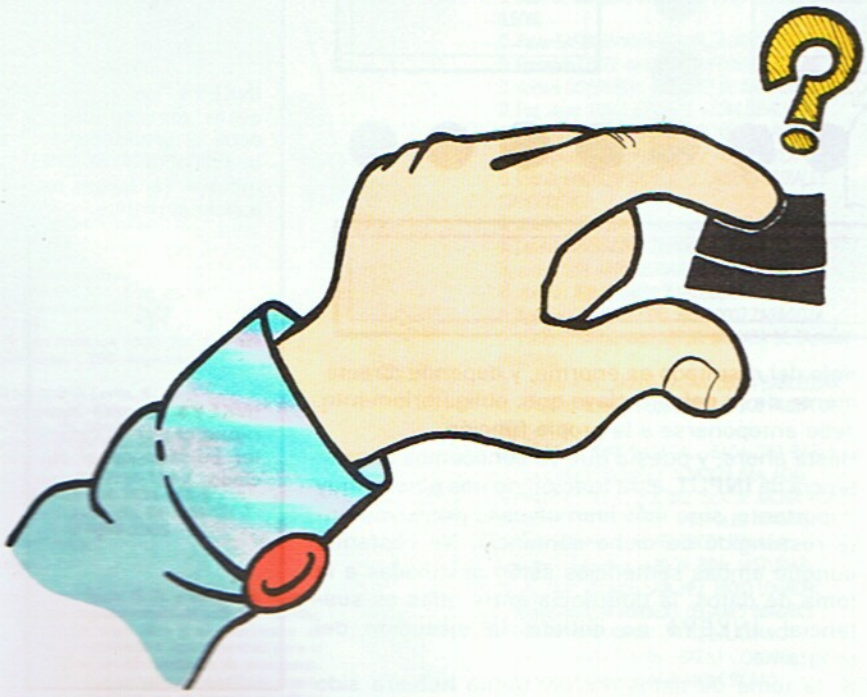
Ahora bien, puesto que dada esa alta velocidad de la que hemos hablado no podemos calcular en qué momento pasa el programa por la sentencia **INKEY$**, ¿qué sucederá si en el momento de su ejecución no existe ninguna tecla pulsada? La respuesta es bien fácil, la función nos lo hará saber tomando por resultado la cadena nula (vacía). Debido a esta facultad, la función **INKEY$** se suele emplear para que se efectúe una detención del programa hasta que pulsemos alguna tecla. Por ejemplo: **10 IF INKEY$=""" THEN GO TO 10.**

Como podemos observar, el programa continuará ejecutando el bucle de la línea 10 siempre y cuando el resultado de **INKEY$** sea la cadena nula, o dicho en otras palabras, mientras que no exista ninguna tecla pulsada.



Mientras que INPUT *detiene la ejecución de los programas,* INKEY$ *efectúa la entrada de datos sin producir este efecto.*

La función INKEY$ *nos permite averiguar el carácter pulsado en el teclado.*

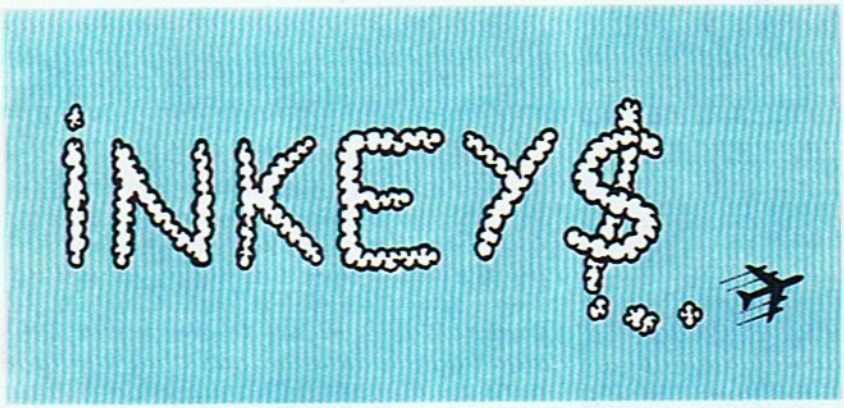


Bien es cierto, que la instrucción **PAUSE 0** conseguiría un efecto parecido, y con menos ocupación de espacio, sin embargo, éste sólo será similar. Tengamos en cuenta que **PAUSE 0** se limita a aguardar la pulsación de cualquier tecla, mientras que **INKEY$** utilizado en la estructura anteriormente estudiada, en combinación con **IF**, además de aguardar la pulsación de una tecla, informa de cual ha sido el carácter entrado.

UTILIDAD DE INKEY$

Su uso es prácticamente obligado en los MENUS, término con el cual se designan los puntos del programa en los cuales éste se detiene en espera de que el usuario seleccione una entre varias opciones. Observemos esto sobre un ejemplo práctico:

```
10 CLS
20 INVERSE 1
30 PRINT "  - MENU DE OPCIONES -  "
40 PRINT ""1"; INVERSE 0;" Opcion uno"
50 PRINT ""2"; INVERSE 0;" Opcion dos"
60 PRINT ""3"; INVERSE 0;" Opcion tres"
70 INVERSE 0
80 IF INKEY$="1" THEN PRINT "SELECCIONADA OPCION UNO": STOP
90 IF INKEY$="2" THEN PRINT "SELECCIONADA OPCION DOS": STOP
100 IF INKEY$="3" THEN PRINT "SELECCIONADA OPCION TRES": STOP
110 GO TO 80
```



La función INKEY$ *se hace prácticamente imprescindible en los juegos de acción.*

En esta ocasión, se han combinado tres estructuras **IF INKEY$**, una detrás de otra, por un sistema que se conoce como CASCADA, de forma que si ninguna de las tres se cumple, es decir, tanto si no se ha pulsado ninguna tecla, como si la tecla no se encontraba dentro del rango de las opciones, se volverá a realizar la toma de datos. Así pues, el hecho de que **INKEY$** realice la entrada de un sólo carácter, tiene la ventaja además de no necesitar la pulsación de **ENTER**, y por tanto, no producir la detención del programa: simplifica enormemente las tareas de depuración.

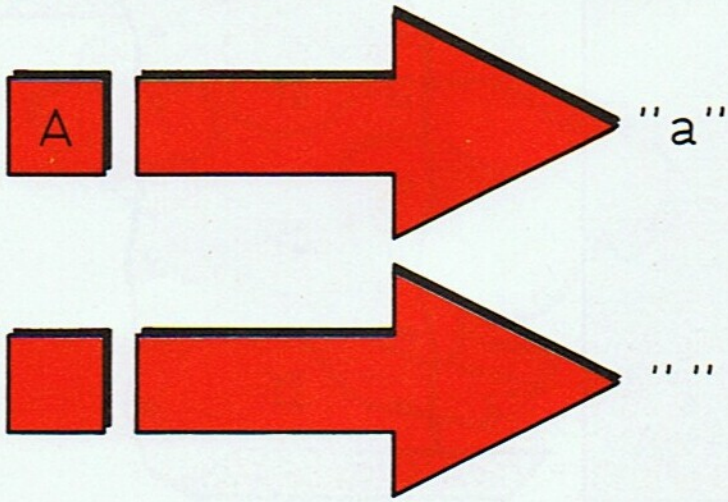
INKEY$ *se utiliza frecuentemente para la confección de menús de programa.*

De hecho, sobre todo cuando el número de opciones es más grande, no se suele utilizar el método expuesto de decisión en cascada, sino que se efectúa una depuración previa de los datos, como podría ser: **80 IF INKEY$<"1" OR INKEY$>"3" THEN GO TO 80.**

Sin embargo, cuando se utiliza **INKEY$** varias veces seguidas, cabe la posibilidad de que, si ha transcurrido un cierto tiempo desde la depuración de la entrada hasta el análisis de la misma, ésta se haya alterado, con el correspondiente error. Veamos un ejemplo.

El siguiente programa, va a multiplicar por 2 un número tomado mediante **INKEY$** y, por tanto, de un sólo dígito. El primer problema será realizar una depuración para que sólo se tomen valores numéricos, y acto seguido, convertir mediante **VAL** el resultado en forma de cadena devuelto por **INKEY$** a número tratable por el operador de multiplicación (*).

Cuando al ejecutarse INKEY$ *no existe ninguna tecla pulsada, se devuelve como resultado la cadena vacía.*



Dado que **INKEY$** efectúa la entrada de un solo carácter, no precisa de la pulsación posterior de **ENTER** como sucede con **INPUT**.

El uso de **INKEY$** es prácticamente obligado en cualquier juego de acción.

La pulsación simultánea de **CAPS SHIFT** y **SYMBOL SHIFT (EXTENDED)** sí que es tenida en cuenta por **INKEY$**, devolviendo el resultado **CHR$ 14**.

```
10 IF INKEY$<"0" OR INKEY$>"9" THEN GO TO 10
20 FOR I=0 TO 499: NEXT I
30 PRINT VAL INKEY$*2
```

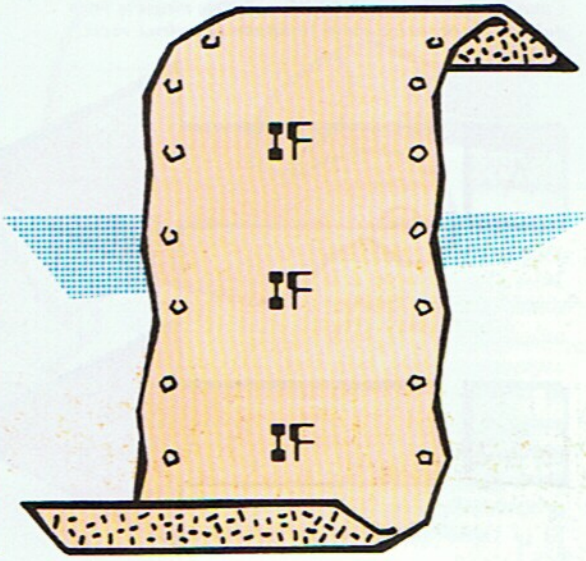
El hecho de que el programa tarde en ejecutarse algún tiempo, se debe al retardo de la línea 20, que ha sido introducido intencionadamente, para producir una diferencia de tiempo entre el momento de la depuración de datos (línea 10), y el de su análisis (línea 30). A menos que hayamos esperado pacientemente, con la tecla pulsada hasta que aparezca el resultado, el programa se detendrá con un mensaje *C Nonsense in BASIC*. ¿A qué es debido ésto? ¿Acaso no ha funcionado el sistema de depuración?

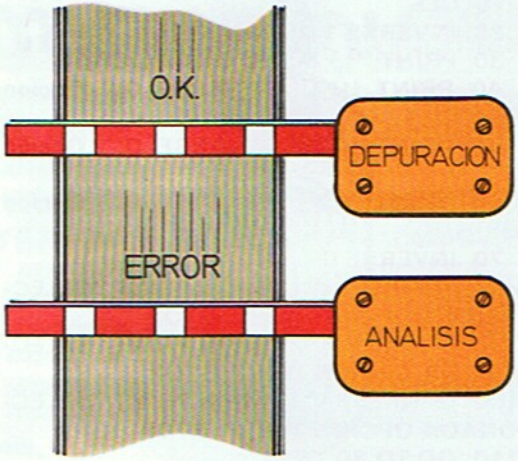
En absoluto. El problema radica en que el valor de **INKEY$** no permanece almacenado, sino que depende del momento de la ejecución de la función, y por tanto, es susceptible de cambios extraordinariamente celéricos. Así, cuando se pulsó una tecla, la línea de depuración funcionó a la perfección y filtró cualquier código que no estuviera comprendido entre cero y nueve.

No obstante, al llegar el programa a la línea 30, el valor de **INKEY$** había pasado a ser nulo, al cesar la pulsación de la tecla, y por tanto se produjo un error cuando intentó efectuarse el **VAL** sobre una cadena vacía.

Para evitar este tipo de inconvenientes, el sistema más frecuente de uso para **INKEY$** es en combinación con la asignación a una variable (**LET**), de forma que el resultado de la función quede registrado, y permanezca constante en todo momento: desde la depuración hasta su análisis. Transformemos el ejemplo anterior.

En las decisiones en cascada, un IF *sucede a otro.*





Con INKEY$ *existe la posibilidad de que entre el punto de depuración de entrada y el de análisis, el resultado se vea alterado.*

```
10 LET X$=INKEY$: IF X$<"0" OR X$>"9" THEN GO TO 10
20 FOR I=0 TO 499: NEXT I
30 PRINT VAL X$*2
```

Un último problema que surge con frecuencia al utilizar la función **INKEY$** es el debido a su extraordinaria rapidez. Ejecutando el siguiente miniprograma, y pulsando cualquier tecla, nos daremos cuenta de ello inmediatamente:

```
10 CLS
20 PRINT INKEY$;
30 GO TO 20
```

Para evitar este efecto de autorrepetición, en muchas ocasiones no deseado, utilizamos una estructura en base a dos **INKEY$**, que facilitan la toma de datos mediante esta función, pero carácter a carácter. Veamos una aplicación práctica de ésto en una nueva versión del último miniprograma:

```
10 CLS
20 IF INKEY$<>"" THEN GO TO 20
30 PRINT INKEY$;
40 GO TO 20
```

En la línea 20 se generan los gráficos correspondientes a la A, B y C, a partir de la configuración decimal expuesta en la **DATA** de la línea 400. En este caso, al ser tres caracteres consecutivos los que componen el *sprite*, podemos permitirnos el «ahorro» de efectuar 24 **POKEs** seguidos, comenzando en la dirección de inicio dada por **USR "A"**.

Esto nos hace llegar a la conclusión de que expresiones tales como **USR "A"+8** y **USR "B"** son sinónimas o **USR "A"+16** y **USR "C"**. Efectivamente, puesto que **USR "<letra>"** no constituye más que la forma abreviada de expresar una dirección, podemos permitirnos utilizarla en la forma que nos convenga, como si de cualquier otro valor numérico se tratara.

Las líneas 30 a 70 efectúan la inicialización general del juego, es decir, ajustan los marcadores a los valores iniciales, dibujan las baterías antiaéreas, etc...

Como podemos observar, el artífice del cambio es la línea 20, la cual actúa como filtro e impide el paso a la impresión hasta que no se ha dejado de pulsar la tecla. La aplicación más concreta de este método, se da con una doble estructura **IF INKEY$**..., la primera de las cuales elimina el efecto de autorrepetición, y la segunda propicia la espera de la pulsación de cualquier tecla.

```
10 IF INKEY$<>"" THEN GO TO 10
20 IF INKEY$="" THE GO TO 20
30 PRINT INKEY$
40 GO TO 10
```

RESULTADO DE INKEY$

Para finalizar con este tema, matizaremos cual es el resultado que devuelve **INKEY$**. Como era de suponer, la función no se ve afectada por todas las teclas, sino que ignora las pulsaciones de aquellas que no equivalen a ningún carácter, como es el caso de **SYMBOL SHIFT** o **CAPS SHIFT**.

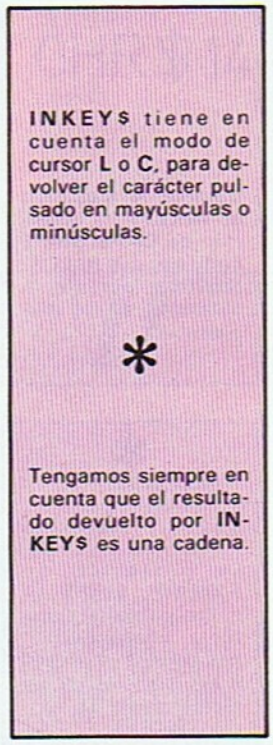
Sin embargo, sí las tiene en cuenta para la composición de caracteres que las precisan, como es el caso de los símbolos que se obtienen mediante **SYMBOL SHIFT** o las mayúsculas, obtenidas pulsando simultáneamente **CAPS SHIFT**.

Así mismo, también se advierte la pulsación simultánea de los dos **SHIFT, CAPS** y **SYMBOL**, generando el resultado **CHR$ 14**, correspondiente al código de separador de número en el código A.S.C.I.I. del Spectrum, pero que también se utiliza para la representación de **EXTENDED MODE**.

Cabe destacar que **INKEY$** se ve igualmente afectado por los modos **C** y **L** del cursor, de forma que el resultado de la función en modo **C**, al

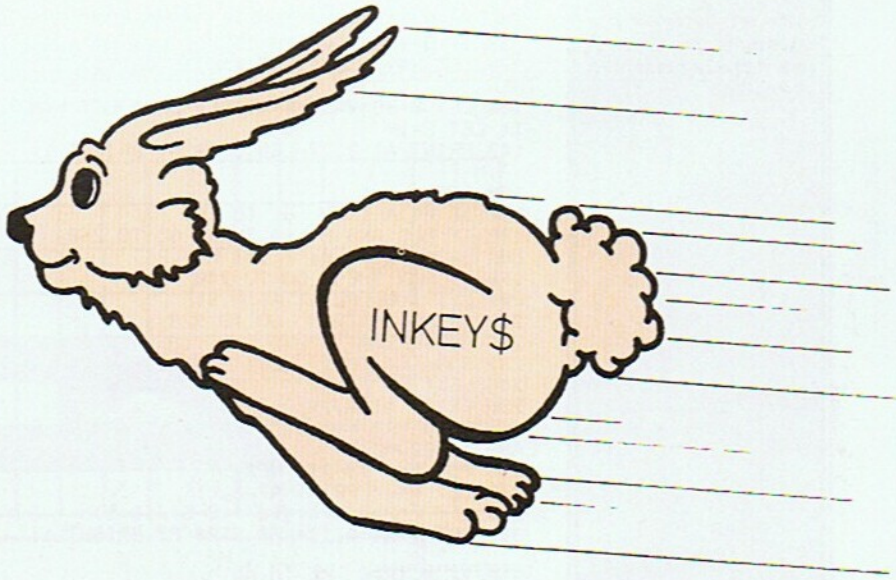
La forma más segura de utilizar INKEY$ *es en combinación con la sentencia* LET, *para almacenar el resultado de la función.*

INKEY$ tiene en cuenta el modo de cursor **L** o **C**, para devolver el carácter pulsado en mayúsculas o minúsculas.



Tengamos siempre en cuenta que el resultado devuelto por **INKEY$** es una cadena.

La gran velocidad de ejecución de INKEY$ *puede convertirse en ciertas ocasiones en un inconveniente.*



Durante la ejecución de **INKEY$**, la pulsación de **CAPS SHIFT** junto con una tecla alfabética, permite la obtención de las mayúsculas.

La pulsación de **SYMBOL SHIFT** simultáneamente a una tecla alfanumérica, nos permite el acceso a los símbolos de las teclas desde la función **INKEY$**.

Por menú se conoce el conjunto de opciones que en determinado momento de un programa se le presentan al usuario.

pulsar cualquier letra, será siempre la mayúscula, sin necesidad de concurrir la pulsación de **CAPS SHIFT**.

Así pues, es importante hacer notar que, como dijimos al comenzar el estudio de esta función, su resultado es el carácter pulsado en el momento de la ejecución y no la tecla, dado que en ocasiones ambas expresiones no son sinónimas. Por ejemplo, con aquellos caracteres que precisan para su obtención de la pulsación de un **SHIFT**.

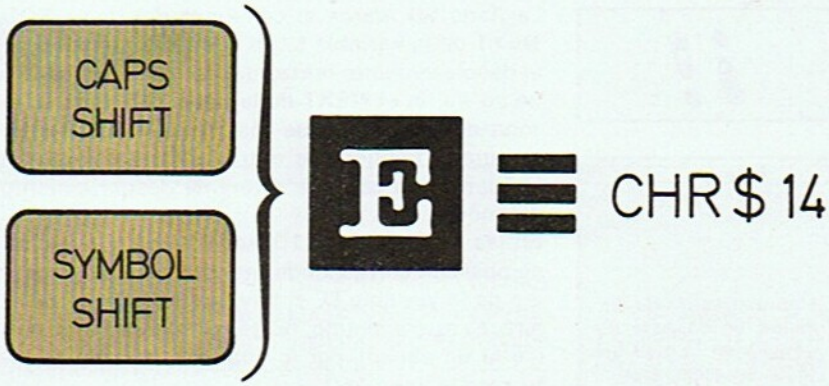
COMIENZA LA ACCION

Como lo prometido es deuda, he aquí el esperado programa anunciado al comienzo de este capítulo, que aplica a un juego la función **INKEY$**, y pone en práctica nuestros conocimientos recientemente adquiridos sobre los atributos de color y la generación de gráficos definibles.

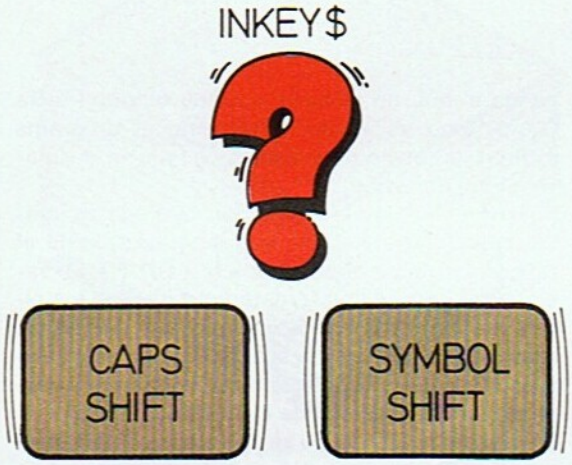
```
  10 REM ANTIAEREO - J.M. LOPEZ MARTINEZ
  20 FOR I=0 TO 23: READ X: POKE USR "A"
+I,X: NEXT I
  30 BORDER 5: PAPER 5: LET R=0
  40 CLS : PRINT AT 21,0; PAPER 6;"88888
818888888882888888888388888"
  50 PRINT AT 20,5;"231";TAB 15;"231";TA
B 25;"231"
  60 PRINT AT 0,0; BRIGHT 1; INK 1;"PUNT
OS..:  0"; BRIGHT 0, BRIGHT 1;"MUNICION:
 50"; BRIGHT 0
  70 LET D=0: LET P=0: LET M=50
  80 LET A=INT (RND*14)+2
  90 FOR L=0 TO 28
 100 PRINT AT A,L; INK 7;" ABC"
 110 IF D THEN  GO TO 150
 120 LET X$=INKEY$: IF X$<"1" OR X$>"3"
THEN  GO TO 230
 130 LET B=10*VAL X$-4: LET D=1: LET M=M
-1: LET H=19
 140 PRINT AT 0,27-LEN STR$ M; BRIGHT 1;
" ";M
 150 LET H=H-1
 160 IF H<>A THEN  GO TO 180
 170 IF B>L AND B<L+4 THEN  GO TO 260
 180 PRINT AT H,B;"!";AT H+1,B;" "
 190 IF H>2 THEN  GO TO 240
 200 LET D=0: PRINT AT H,B;" "
 210 IF NOT M THEN  GO TO 320
 220 GO TO 240
 230 FOR T=0 TO 1: NEXT T
 240 NEXT L
 250 PRINT AT A,29;"   ": GO TO 80
 260 PRINT AT H+1,B;" ";AT A,L; INK 2; F
LASH 1;"BOUM"
 270 FOR T=0 TO 99: NEXT T
 280 LET D=0: PRINT AT A,L;"    "
 290 LET P=P+(16-A)*(4-((B-6)/10+1))
 300 PRINT AT 0,12-LEN STR$ P; BRIGHT 1;
P
 310 IF M THEN  GO TO 80
 320 IF P<=R THEN  GO TO 350
 330 PRINT AT 9,0;"HAS ESTABLECIDO UN NU
EVO RECORD!"
 340 LET R=P: GO TO 360
 350 PRINT AT 9,9;"EL RECORD SIGUE"
 360 PRINT AT 11,16-(10+LEN STR$ R)/2;"E
N "; BRIGHT 1;R; BRIGHT 0;" PUNTOS"
 370 PRINT AT 13,2;"DESEAS INTENTARLO DE
 NUEVO ?"
 380 LET X$=INKEY$: IF X$="S" OR X$="s"
THEN  GO TO 40
 390 IF X$<>"N" AND X$<>"n" THEN  GO TO
380
 400 DATA 0,192,224,240,255,159,127,0,0,
0,7,15,255,192,255,0,0,0,128,192,248,255
,248,0
```

La estructura general del programa antiaéreo es de una extraordinaria simplicidad.

A partir de este momento, comenzamos a hacer uso del color, asignando tanto al fondo de la pantalla (**PAPER**) como al marco (**BORDER**) un color azul celeste, con el cual simular el despejado cielo, escenario de la inminente batalla.
La línea 40 incorpora un **CLS** que hace extensivo el atributo de color **PAPER** a toda la pantalla. En este caso, las sentencias de color presentes en la línea 30 y la de la 40 (*CLS*), se han separado, cosa aparentemente ilógica. Esto se debe a que 40 marca la división entre las inicializaciones de programa y las de juego.
Esto quiere decir que las líneas de la 10 a la 30 sólo son ejecutadas una vez, al lanzarse por primera vez el programa. Estas son las llamadas inicializaciones de programa, y fijan valores que permaneceran inalterables hasta finalizar el mismo, tales como las definiciones de gráficos, los colores principales del juego (fondo y marco), y el record, almacenado en la variable **R**.
En esta zona de programa se encuentran todos los gráficos predefinidos. Así pues, los caracteres doblemente subrayados de las líneas 40 y 50 deben ser introducidos como los gráficos cambiados de las teclas afectadas; es decir, los que se obtienen por la pulsación simultánea de **CAPS SHIFT** y la tecla correspondiente en el modo **GRAPHICS (CAPS SHIFT + 9)**.
En la línea 80 se genera de forma aleatoria la altura del avión para cada nueva pasada por la pantalla. Como es evidente, se le han marcado unos límites, de forma que no vuele tan alto como para borrar a su paso los marcadores de la línea superior, ni tan bajo que sea demasiado fácil de derribar o choque con las posiciones de los antiaéreos.
Es en este punto en el cual comienza lo que podríamos llamar ciclo principal del programa, puesto que es dentro de él en el cual discurre el juego, hasta que se termina la munición.

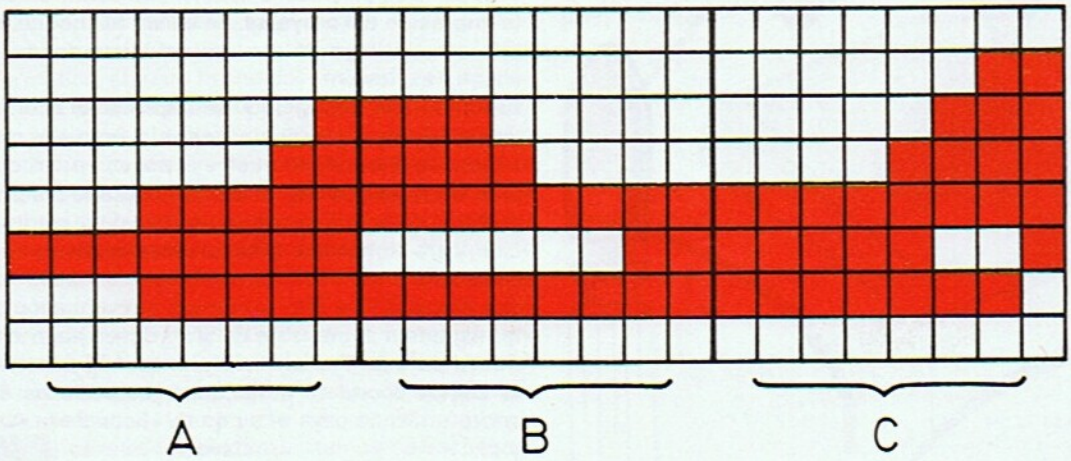


Para INKEY$ *el cursor E (*EXTENDED*) equivale a un* CHR$ 14.



La pulsación independiente de CAPS SHIFT *y* SYMBOL SHIFT *es ignorada por* INKEY$.

El sprite *utilizado en el programa antiaéreo está formado por los gráficos definidos de las teclas A, B y C.*



Entre el punto de depuración de la entrada por **INKEY$** y su análisis, puede que se produzca una alteración del resultado de la función.

Para evitar posibles errores por variaciones de resultado, el mejor sistema para utilizar **INKEY$** es en combinación con una sentencia de asignación **LET**.

¡!

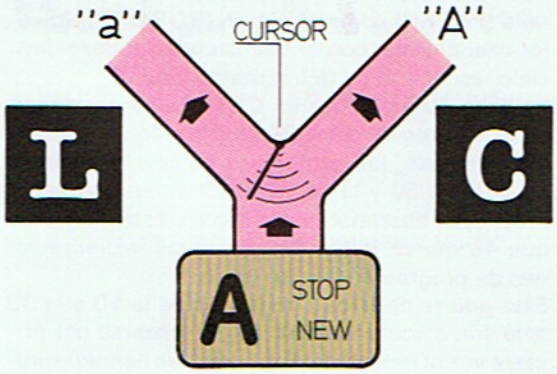
Una estructura de decisión en cascada es aquella en la cual los **IF** se suceden, produciendo bifurcaciones por distintas condiciones.

La línea 90 marca el comienzo del bucle **FOR NEXT** de la variable **L**, en el cual se comprende el desplazamiento horizontal del avión, y que tiene su fin en el **NEXT** de la línea 240. Esta es la zona de programa que más frecuentemente se ejecuta, y contiene la entrada y tratamiento de los datos, así como la rutina de desplazamiento del móvil.

En las líneas 100 y 110, se imprime el avión en su posición correspondiente y se evalúa, por medio de la variable **D**, si hay algún proyectil en el aire. Si existe alguno, no se permite disparar otro (sería un abuso), por lo cual el programa se bifurca a la línea 150.

Tengamos en cuenta a la hora de introducir el programa, que el *sprite* del avión está construido con caracteres gráficos, y por tanto, las letras ABC subrayadas, que aparecen en la línea 100 (impresión del avión), deben ser sustituidas por los caracteres gráficos de las teclas afectadas.

En la línea 120 se aceptan caracteres del teclado por medio de la sentencia **INKEY$**. Como los disparos se producen pulsando las teclas del 1, 2 y 3, está claro que cualquier otro valor, incluyendo el que no se haya pulsado ninguna tecla (**X$=""**) no es válido, bifurcando el programa al final del bucle de **L** para proseguir el desplazamiento del avión.



El resultado de **INKEY$** *se ve afectado por los modos* **C** *y* **L** *del cursor.*

Fijémonos en que la desviación se dirige concretamente a la línea 230, que sirve de retardo al programa, empleando un bucle **FOR-NEXT** vacío, para igualar la diferente velocidad de los ciclos de programa en los cuáles no se pulsan teclas válidas, puesto que se efectúan en este caso menos comparaciones con el consiguiente incremento en la velocidad de ejecución.

En las líneas 130 a 150 se calcula la posible puntuación (**B**); en proporción a la batería que efectúa el disparo, se conecta el indicador de proyectil en el aire (**D**), se resta un proyectil, y se establece el valor inicial de desplazamiento vertical de este, actualizándose el marcador de munición.

En las líneas 160 y 230, se evalúa si ha existido colisión entre el proyectil y el avión, comparando los valores de las ordenadas y abcisas respectivas. Si no se produce la colisión, se continúa con la impresión del proyectil; de existir el choque, se bifurca a la línea 260, fuera del bucle donde se trata la explosión.

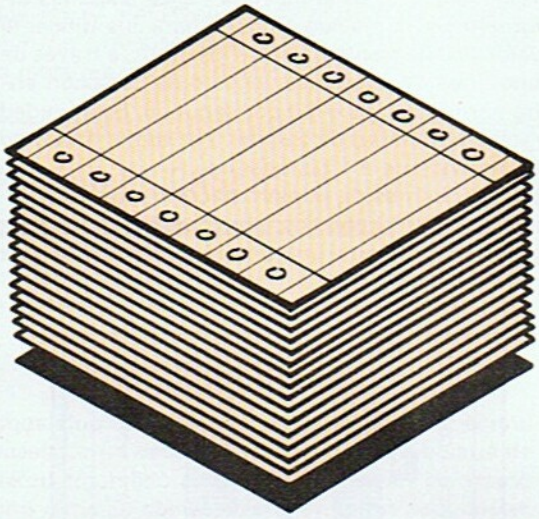
En las líneas 300 y 360, se imprime el acumulado de puntos, y se comprueba si aún queda munición; de ser así, se vuelve a poner en circulación un nuevo avión. Si se han agotado los proyectiles, se prosigue con el análisis de la puntuación, para comprobar si se ha establecido un nuevo record.

Las líneas 370 a 390 contienen la evaluación de la respuesta al mensaje de si se desea jugar una nueva partida. Por último, la línea 400 contiene la **DATA** donde se almacenan los *patterns* del *sprite* utilizado para el avión (gráficos de la A, B y C).

Las líneas 160 a 230 del programa antiaéreo son las encargadas de comprobar si el proyectil ha hecho blanco.

CASETE SI, CASETE NO

EL almacenamiento masivo de datos ha constituido de por sí uno de los caballos de batalla más traídos y llevados entre los fabricantes de micros en los últimos años.



Por el módico precio de una C-60 podemos almacenar de una manera fiable unos 640 Kb. de información.

No sólo por la guerra de *software* que desencadena, sino por todas aquellas cuestiones referentes a rapidez, fiabilidad y precio, que van parejas al soporte utilizado como medio de almacenamiento externo de la información. Sin duda, el más popular de todos, es el casete.

Quizás, por parte de los profesionales de la informática, el mero hecho de mencionar en su presencia la palabra casete, haga aparecer una mueca de horror o desagrado en sus rostros. Incluso un pequeño ataque de nervios puede llevar a la desesperación al técnico, que por exigencias del trabajo, necesite de acceso y transmisión de datos a elevadas velocidades, si tan solo dispone de un casete de audio convencional para efectuar su labor.

Pero los usuarios del Spectrum, al menos en una etapa inicial, ponemos mucha más confianza en las posibilidades que como medio de acopio masivo de datos y programas nos puede proporcionar el casete. No obstante, hemos de reconocer algunas deficiencias del sistema, pero veremos como también otras muchas virtudes lo hacen especialmente útil, incluso insustituible, en función del trabajo que deseemos realizar.

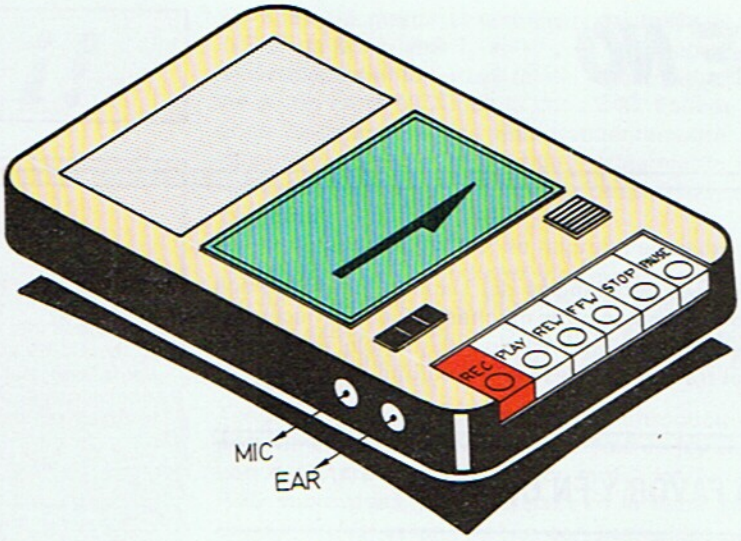
A FAVOR Y EN CONTRA

Tres son los factores primordiales que condicionan la utilidad de cualquier periférico como soporte exterior de información:

1. Tiempo de acceso a la información.
2. Tiempo de transmisión de ésta desde o hacia el ordenador.
3. Fiabilidad del proceso.

Obviamente, el mejor sistema sería aquel que consiguiera localizar la información casi instan-

El casete es el medio de almacenamiento masivo más utilizado en los microordenadores.



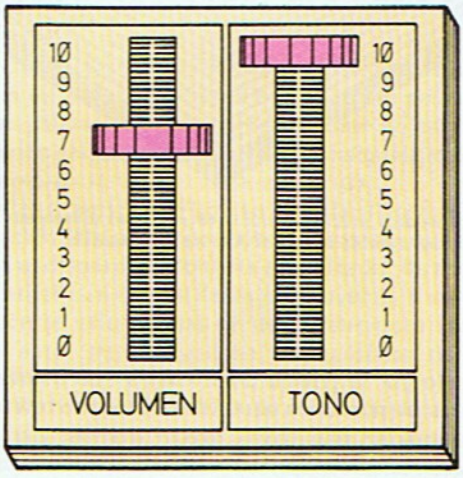
Los pequeños casetes del tipo «periodista» suelen dar muy buenos resultados con un bajo precio.

táneamente, enviarla o recibirla en el menor tiempo posible, y además, no dejara lugar a errores de transmisión (máxima fiabilidad).
Desgraciadamente para nosotros, disponer de un equipo que conjugue óptimamente las tres posibilidades, conduce directamente a un cuarto factor, tan importante como los tres anteriores: el precio. Este será tanto más elevado, cuanto más aumenten la velocidad y seguridad en los procesos de transferencia de datos que involucren a nuestro periférico encargado de gestionar el trasvase de la información.
Precisamente el precio es una de las mayores virtudes de los casetes. Por un lado, no necesitaremos un modernísimo equipo de alta fidelidad para poder usarlo con nuestro Spectrum. Bastará una pequeña grabadora portatil, a la cual exigiremos un mínimo de cualidades que comentaremos más adelante. En la actualidad, por menos de diez mil pesetas, pueden conseguirse en el mercado aparatos que cumplan a la perfección su cometido.
Por otra parte, el medio utilizado como soporte de la información, lo que popularmente se conoce como cinta casete, es con diferencia el más barato entre los utilizados para el almacenamiento masivo de datos. Baste decir como ejemplo, que una cinta C-60, de una hora de duración (treinta minutos por cada cara), con un precio medio en torno a las doscientas pesetas, es capaz de almacenar del orden de 640 Kbytes.

Las pilas suelen gastar malas pasadas en el momento más inoportuno; conviene que nuestro casete disponga de un sistema de alimentación por red.

VELOCIDAD Y FIABILIDAD

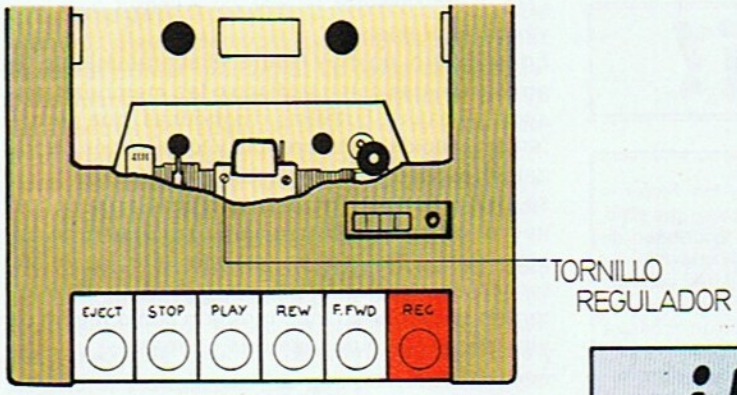
Muchos de los usuarios se quejan de la extremada lentitud, cuando se trata de grabar o recuperar programas almacenados en cinta. Bien, es cierto; y además, esos tiempos de espera pueden llegar a exasperar al más paciente. Estudiemos algo más a fondo este problema.
Técnicamente, en la mayoría de los sistemas informáticos, la medida utilizada para cuantificar la velocidad de transmisión de los datos, a través de una línea de comunicaciones o de conexión entre periféricos, es el bit por segundo. Esta unidad recibe el nombre de Baudio. Por tanto, 1 baudio es equivalente a 1 bit/segundo.
Los fabricantes de nuestro micro fijaron esta velocidad a 1500 baudios, aproximadamente. Y alguien seguramente preguntará: ¿y por qué no una más elevada?



Si el casete no dispone de ajustes automáticos de volumen y tono, es conveniente situar el primero a tres cuartos de la escala, y el segundo al máximo de agudos.

La respuesta es muy sencilla: es fiable. Es decir, asegura que el ordenador «no se hará un lío» al interpretar la información que recibe desde la grabadora, ni que la cinta magnética llegue a saturarse, de manera que los datos allí almacenados resulten irreconocibles, para el Spectrum, durante el proceso de carga.
No obstante, las rutinas que gestionan todo lo relacionado con la grabación y carga de datos en casete, están albergadas en la R.O.M., es decir,

el control del proceso se realiza por *software*. Desde luego, el contenido de la memoria de sólo lectura (R.O.M.), como sabemos, es inalterable, a menos que nos empeñemos, por medios más o menos violentos, y nada aconsejables, en tratar de modificarla.
Obrando con cierta habilidad, y utilizando técnicas en absoluto perjudiciales para la integridad física de la R.O.M. del Sistema, veremos en un próximo capítulo como la velocidad de transmisión puede ser elevada a 2000, 2500, 3000 baudios... e incluso más. O podremos disminuirla, con el sano objetivo de obtener copias de seguridad, con una calidad elevada.
Seguirá habiendo quien piense que el casete es lento, comparado con otros periféricos empleados en el almacenamiento masivo de información, pero quedará demostrado que no tanto como la mala prensa le atribuye.



Un mal alineamiento de la cabeza de lectura/escritura (azimut) puede ser la causa de problemas en la lectura de programas grabados con otros casetes o adquiridos por canales comerciales.

ACCESO EN CASETE

Cuestión muy diferente es la relacionada con el acceso a un determinado programa o bloque de datos contenido dentro de una cinta. No queda otro remedio que realizarlo de forma secuencial, avanzando y retrocediendo la cinta, actuando sobre las teclas adecuadas, hasta localizar el fragmento deseado. Todo ello contribuye a elevar el tiempo de acceso, sobre todo si se trata de cintas de larga duración.
En la actualidad, aparecen en el mercado algunos casetes especiales que llevan incorporado un sistema de busqueda rápida de programas, lo cual supone una considerable ventaja, si tenemos en cuenta que lo normal es tener almacenados varios bloques de información dentro de la misma cinta.
Es precisamente este factor el que los detractores de los casetes argumentan, no exentos de razón, como «falta de operatividad». Consideremos, por ejemplo, un fichero en el cual tuviéramos almacenados el nombre de nuestros amigos y sus respectivos números de teléfono. Si alguna de nuestras amistades cambia de número, no tendremos más remedio que extraer de la cinta el fichero completo, modificar en la memoria el contenido de la variable que conservava el número antiguo, y volver a grabar en cinta el nuevo fichero.
Por tanto, la posibilidad de disponer de ficheros de acceso directo, los cuales permiten la modificación o lectura de un dato específico, sin necesidad de leer el archivo completo, está vedada si el soporte utilizado es la cinta magnética.
La facilidad de manejo de la grabadora es otra de las razones por las cuales son tan populares los sistemas a casete, dentro del campo de los microordenadores. Aunque no debemos olvidar que las operaciones a realizar en el equipo para su puesta en funcionamiento, tanto para carga, grabación o búsqueda de programas, deben ejecutarse manualmente, mientras que en otros sistemas, como el microdrive o la unidad de disco, es automática.
Muchos de los usuarios se quejan de los problemas que surgen al tratar de cargar un programa almacenado en una cinta, grabada en distinto equipo del propio. Normalmente, son achacables al deficiente ajuste de la cabeza de lectura/escritura. Pero más adelante veremos como esto no es óbice para el perfecto funcionamiento del ca-

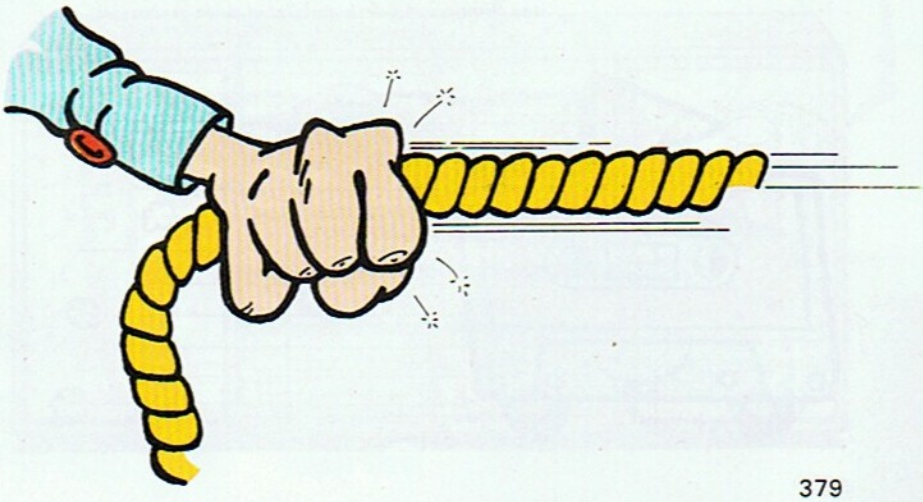
¡!

Las rutinas que gestionan todo lo relacionado con la grabación y la carga de datos en casete, están albergadas en la R.O.M.; es decir, el control de estos procesos se realiza por el *firmware*.

*

En la actualidad aparecen en el mercado algunos casetes especiales, que incorporan sistemas de búsqueda automática de programas.

El mecanismo de arrastre de la grabadora es muy importante para la lectura/escritura de programas.



¡!

Tres son los factores primordiales, que condicionan la utilidad de cualquier periférico como soporte exterior de información:
1. Tiempo de acceso a la información.
2. Tiempo de transmisión de ésta desde o hacia el ordenador.
3. Fiabilidad del proceso.

*

El precio es una de las mayores virtudes de los casetes. No necesitaremos un modernísimo equipo de alta fidelidad, bastará una pequeña grabadora portátil.

sete, e indicaremos el método a seguir para solventar el problema.
En un futuro, no muy lejano, el abaratamiento de otros sistemas como el disco o las memorias masivas a base de burbujas magnéticas, que permiten una rápida transferencia y localización de los datos, desplazarán al casete, relegándole a un segundo término, donde su cometido no será otro que el de conservar copias de seguridad de los programas y ficheros de uso corriente. Hasta entonces, la relación precio-capacidad, lo convierten en el periférico más utilizado como medio de almacenamiento de memoria en masa.

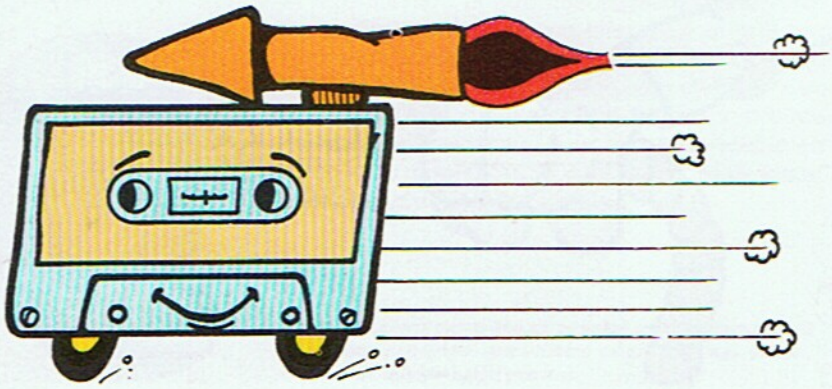
UNA GRABADORA IDEAL

Los enemigos acérrimos del uso del casete como soporte de información, basan sus quejas en los excesivos problemas y elevado porcentaje de errores que pueden producirse durante las labores de lectura/escritura.
En efecto, estos son los momentos criticos, pero la elección adecuada, tanto de la cinta como de la grabadora, puede disminuir el tanto por ciento de error a límites bajísimos.
La idea de conectar nuestro Spectrum a un modernísimo equipo de alta fidelidad, debe ser desestimada. No sólo por problemas de espacio, sino también porque el elevado número de mecanismos destinados a filtrar la señal, atenuarla, ecualizarla, disminuir el nivel de ruido o seleccionar las frecuencias adecuadas para la escucha, tienden a complicar en exceso el manejo. Y todo ello, sin contar con el factor precio, lógicamente más elevado en estos equipos.

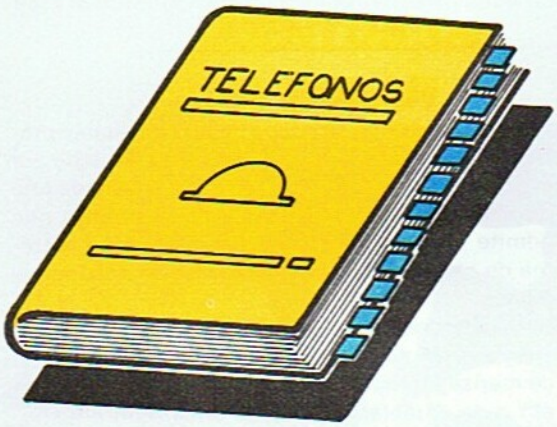
No cabe duda de que la velocidad de transferencia del casete deja bastante que desear.

Los pequeños casetes portátiles, ya sea del tipo utilizado por los periodistas o cualquier otro modelo, suelen ofrecer las suficientes prestaciones para trabajar con nuestro micro. Dentro de éstos, la elección debería ir encaminada hacia uno mono, frente al estéreo. Si bien los últimos ofrecen mejores resultados cuando se trata de reproducir música, la utilización de dos pistas durante el proceso de grabación (grabación estéreo), tiende a provocar problemas de reproducción al menor desajuste de los cabezales de lectura/escritura.
Además del botón PLAY, empleado para la lectura, y del RECORD o REC, que pulsado junto con el anterior, pone al aparato en situación de escritura, serán necesarios como mínimo otros tres pulsadores. Uno de STOP, para detener la grabadora, y otros dos para facilitar el avance (F-FWD, *Fast ForWarD*) y rebobinado (RWD, *ReWinD*) rápido de la cinta. Algunos modelos, permiten el uso conjunto de PLAY+FFWD y PLAY+RWD, con el objeto de controlar mediante el altavoz o los auriculares, la diferencia de sonido entre las zonas grabadas y en blanco, y así detectar el posible comienzo de un programa.
La tecla de pausa no es del todo necesaria, aunque sí muy útil. Permite detener la marcha de la grabadora durante el trabajo de lectura/escritura, sin necesidad de estar constantemente parando y volviendo a poner en marcha el aparato con el uso de PLAY y STOP. El control de tono también puede ser accesorio, y caso de tenerlo, debemos situarlo hacia el máximo de agudos.

El aumento en la velocidad de transferencia conduce a una disminución de la fiabilidad de la grabación.



Resulta imprescindible un sistema de ajuste del volumen durante las operaciones de lectura. Desgraciadamente, no todas las casetes están grabadas al mismo volumen, y será preciso actuar sobre este control para ajustar el adecuado (normalmente, hacia los tres cuartos del máximo). Por lo general, las grabadoras portátiles incorporan control automático del nivel de grabación, que resulta normalmente adecuado a las exigencias de nuestro Spectrum. Actualmente, los denominados «casetes especiales para ordenador», suelen incorporar un sistema automático de volumen, tanto para la lectura como para la grabación.

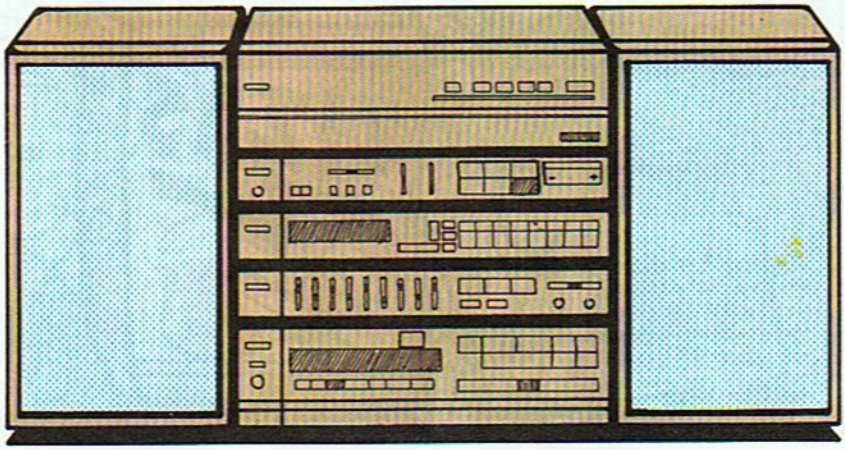


Los archivos de acceso directo están vedados al casete.

El cuentavueltas de cinta constituye una inestimable ayuda a la hora de buscar con mayor rapidez un programa determinado. Conservar las cintas ordenadas, indicando en cada una de ellas los programas que contiene, junto con los datos referentes a longitud, dirección de comienzo (ver programa INDEX) y número de vuelta en el que está ubicado, nos ahorrará innecesarias pérdidas de tiempo al tratar de localizar un programa.
El uso de pilas en grabadoras portátiles está generalizado. Pero es muy importante que el aparato disponga de una salida para poder conectarlo directamente a la red, a través de una pequeña fuente de alimentación. La experiencia demuestra que las pilas tienen la mala costumbre de gastarse en el momento más inoportuno, ocasionando irregularidades en el desplazamiento de la cinta, y como consecuencia, problemas en la transmisión de los datos.
La salida EAR y la entrada MIC deben tener el mismo diámetro (3,5 mm), que las clavijas de los cables suministrados con el Spectrum. Algunos aparatos llevan una tercera salida, denominada REMOTE o REM, utilizada para controlar desde el ordenador el motor del casete, pero que con el Spectrum carece de utilidad.

El sistema de arrastre debe ser de calidad. De otra manera, inevitablemente tendremos problemas desde el primer momento. La velocidad de desplazamiento de la cinta está normalizada a 4,76 centímetros por segundo. Por ello, lo más recomendable es adquirir un equipo de algún fabricante de contrastada calidad, y no dejarse engañar por la afirmación: «Cuanto peor sea el casete, mejor cargará el Spectrum».
Los casetes de nueva aparición, que los fabricantes denominan especiales para ordenador, habitualmente no son otra cosa que aparatos normales dotados de volumen constante para lectura y escritura, y un dispositivo que permite mantener simultáneamente conectadas las clavijas EAR y MIC. Tendremos que considerar si merece la pena pagar bastante más que por un equipo normal.
Sobre la cabeza de lectura/escritura, algunos casetes poseen un pequeño orificio, bajo el cual se encuentra un tornillo, destinado al ajuste de la inclinación de la cabeza. Esta debe quedar perpendicular a la cinta, y es asombrosa la facilidad con que se desajusta, dando lugar a problemas durante la carga. Con la grabadora en posición PLAY, y utilizando una cinta de las que contienen información para el Spectrum, podemos girar lentamente el tornillo, mediante un destornillador adecuado, hasta escuchar el sonido lo más estridente y agudo posible.
Esta será la posición de ajuste adecuada, aunque si seguimos teniendo problemas por este motivo, lo mejor es acudir a un servicio técnico especializado. A partir de ese momento, la grabadora comenzará a funcionar mejor; pero, seguramente, los programas contenidos en otras cintas grabadas anteriormente, con las cabezas desajustadas, nos den algún problema a la hora de intentar recuperarlos.

La idea de conectar nuestro Spectrum a un modernísimo equipo de alta fidelidad debe ser desestimada.



¡!

Una cinta de una hora de duración es capaz de almacenar del orden de 640 Kb:

*

La medida utilizada para cuantificar la velocidad de transmisión de los datos es el Baudio, y equivale a un bit por segundo. La velocidad media de transmisión del Spectrum son 1500 Baudios.

*

Utilizando técnicas en absoluto perjudiciales para la R.O.M. del Sistema, podemos alterar la velocidad de transmisión.

EL PROFESOR

LOS ordenadores ayudan al ser humano en multitud de campos, tales como el científico, comercial, estadístico, etc... En nuestro caso, los microordenadores, donde manifiestan una particular habilidad es en la enseñanza de las tareas automáticas, fundamentalmente a los más pequeños.

Son muchos los países que han adoptado este método de enseñanza: Japón, Estados Unidos, Inglaterra. La mente abierta de los jóvenes encuentra en el ordenador un amigo que les ayuda y aconseja en sus tareas. Además, y puesto que irremisiblemente estas máquinas «inteligentes» serán, con toda certeza, parte importante de su futuro, no está de más que desde muy temprana edad se acostumbren y familiaricen con ellas, y comprendan que los ordenadores no son más que aparatos inútiles si no existe un humano que los maneje.

La programación orientada a la enseñanza es sumamente sencilla, aunque no exenta de alguna que otra complicación mínima. En nuestro caso, hemos realizado un pequeño programa, que sirve como botón de muestra de la inmensa cantidad de aplicaciones desarrolladas en este área.

Gracias a este programa, cambiamos la clásica tiza por el teclado del ordenador.

¡!

Cada uno de los sumandos que compongan la operación de adición no ha de exceder de diecinueve dígitos.

*

Una vez introducidas todas y cada una de las líneas de instrucción del programa, podremos grabarlo utilizando el siguiente comando: **SAVE "PROFESOR"**. Si optásemos por la grabación con autoejecución, tecleríamos la instrucción **SAVE "PROFESOR" LINE 10.**

*

Los programas educativos deben encontrarse especialmente protegidos contra posibles circunstancias de error.

EL PROGRAMA

Nuestro programa se ocupa de un simplísimo problema aritmético: la suma. Este está limitado por el número de sumandos y la longitud de cada uno de ellos. La cantidad máxima de sumandos que admite el programa es diez, y la longitud extrema de cada uno, no ha de exceder de diecinueve dígitos.

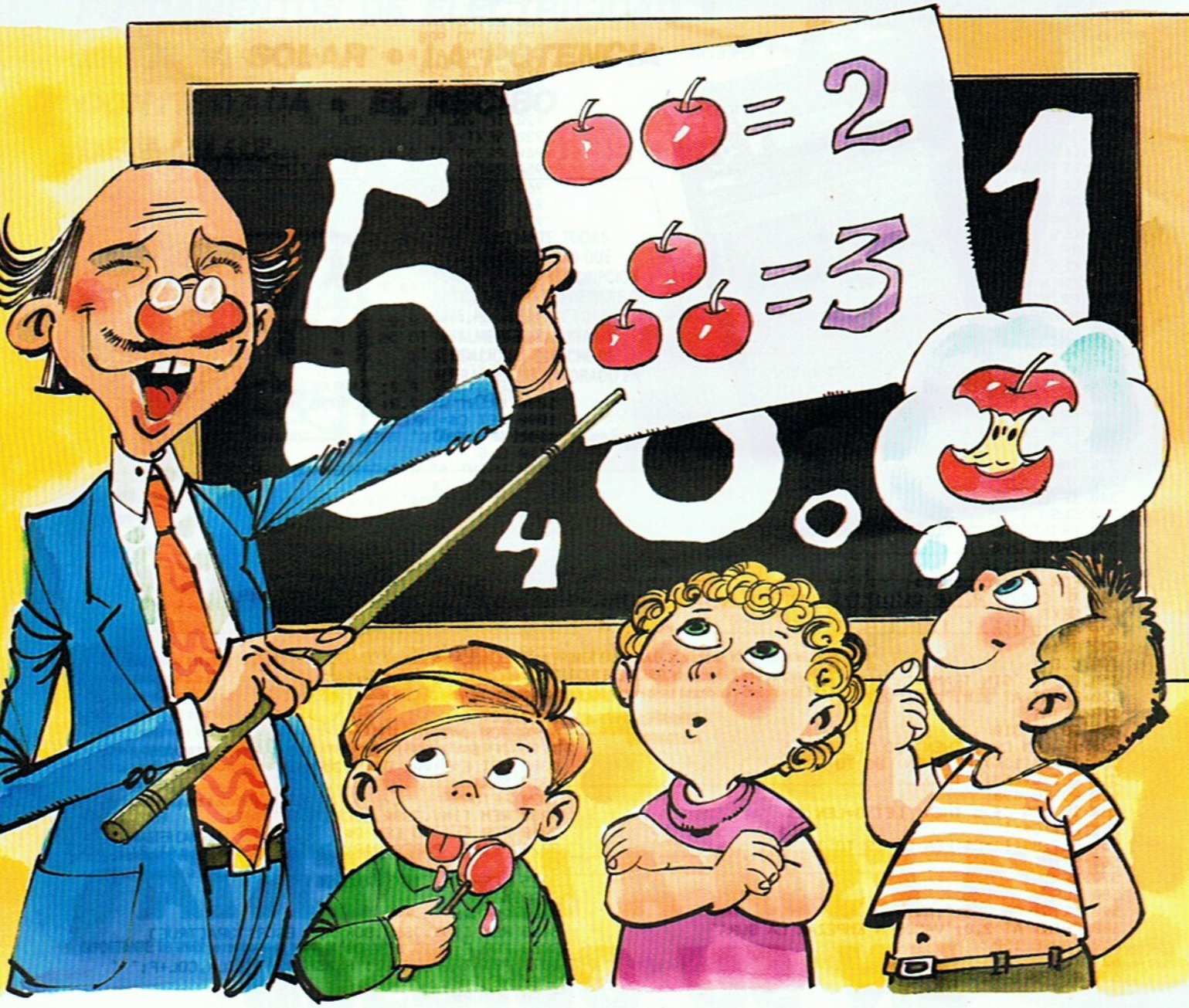
El listado de este programa se podía haber reducido a unas pocas líneas, pero la gran cantidad de mensajes explicativos que se han incluido, con el fin de entablar un diálogo eficaz con el usuario, hacen que la extensión del programa sea algo más acentuada. Cuanto menor es la edad del usuario, mayores deben de ser las precauciones que tome el programador para la correcta asimilación y comprensión de los conceptos que utiliza el programa.

Los programadores debemos tener siempre en cuenta, que un programa es tanto mejor, cuanto menos precise del conocimiento informático del usuario. En este sentido, los programas destinados a los jóvenes, más que ninguno, deben contemplar todas las posibles circunstancias de error que puedan concurrir, de forma que en ningún momento el joven usuario se vea desprotegido o impotente ante la máquina.

Así mismo, es conveniente que el ordenador entable un diálogo lo más ágil posible con el estudiante, de forma que éste llegue a considerarlo más como un compañero o un amigo mayor, que como una simple máquina. Este tipo de relación usuario-ordenador, se conoce como método CONVERSACIONAL, y debe de estar presente en todos los programas educativos como el que hoy presentamos, auxiliando punto por punto al destinatario del programa.

De no cumplirse estas normas, lo más seguro es que todos nuestros esfuerzos por crear un buen programa educativo resulten infructuosos, aunque hayamos empleado una gran cantidad de tiempo y conocimientos en el componente técnico del mismo.

Como es habitual, para la grabación del programa utilizaremos el comando **SAVE**, con la forma **SAVE "PROFESOR"** o bien **SAVE "PROFESOR" LINE 10**, si deseamos que el programa se autoejecute nada más cargarlo.
Para la introducción del listado debemos tener en cuenta que las palabras **INV** y **TRUE** subrayadas y entre corchetes, deben ser sustituidas, respectivamente, por los caracteres **INVERSE VIDEO (CAPS SHIFT + 4)** y **TRUE VIDEO (CAPS SHIFT + 3)**.



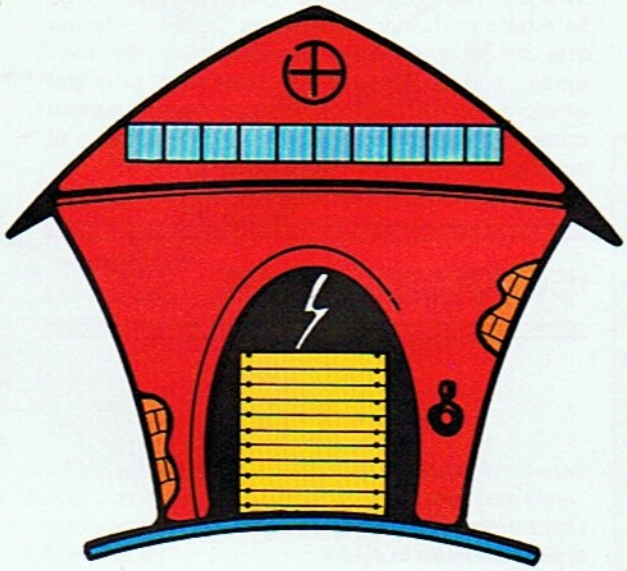
```
   1 REM ************************
   2 REM * J.M.MAYORAL SERRANO *
   3 REM ************************
   4 REM * PROFESOR      1985  *
   5 REM ************************
  10 PAPER 1: INK 9: POKE 23658,8
  20 BORDER 1: CLS
  21 LET PAP=6: LET K=9
  22 LET FL=0
  23 LET FILA=12: LET COL=0
  30 LET Q$='HOLA! SOY TU AYUDANTE DE SUMAS': GO SUB
3000
  31 PAUSE 50
  50 PRINT AT 14,5; PAPER 2;'    QUIEN ERES TU ?    '
  60 INPUT A$
  70 CLS
  71 LET COL=3
  80 LET Q$='  PERFECTO  '+A$+'  '
  81 LET PAP=1: LET FL=1
  82 GO SUB 3000
  83 FOR N=1 TO 30
  84 BEEP .1,INT (RND*3)+40
  85 NEXT N
  86 GO SUB 3050
  90 PRINT  PAPER 5;AT 2,6;'  VAMOS A EMPEZAR  '
 100 PRINT
 110 PRINT 'DIME CUANTOS SUMANDOS TIENE TU'
 130 PRINT 'PROBLEMA'
 140 INPUT A
 150 IF A>1 THEN  GO TO 190
 160 CLS
 170 LET Q$='ESTO ES MUY SERIO: NO BROMEES'
 171 LET FILA=20: LET COL=1
 172 GO SUB 3000
 175 PAUSE 50
 180 CLS : GO TO 100
 190 IF A<11 THEN  GO TO 250
 200 CLS
 210 PRINT 'LO SIENTO, NO PUEDO AYUDARTE EN'
 220 PRINT AT 2,0;'UNA SUMA TAN LARGA'
 230 PAUSE 250
 240 GO TO 1110
 250 CLS
 260 LET B=0
 270 DIM B$(A,20)
 280 PRINT 'DE ACUERDO ';A$
 290 PRINT
 300 FOR I=1 TO A
 310 PRINT 'ESCRIBE EL SUMANDO ';I
 320 INPUT C$
 330 IF C$='' THEN  GO TO 320
 340 IF LEN C$<20 THEN  GO TO 390
 350 CLS
 360 PRINT 'NO SE SUMAR CIFRAS TAN LARGAS'
 370 PAUSE 200
 380 GO TO 1110
 390 FOR J=1 TO LEN C$
 400 IF C$(J)<'0' OR C$(J)>'9' THEN  GO TO 440
 410 NEXT J
 420 GO TO 490
 430 PRINT
 440 CLS
 450 PRINT 'SIN TRAMPAS: SOLO NUMEROS'
 460 PRINT AT 3,0;'OTRA VEZ'
 470 PRINT
 480 GO TO 310
 490 IF LEN C$=1 THEN  GO TO 530
 500 IF C$(1)<>'0' THEN  GO TO 530
 510 LET C$=C$(2 TO )
 520 GO TO 490
 530 IF LEN C$>B THEN  LET B=LEN C$
 540 LET C$='                    '+C$
 550 LET B$(I)=C$(LEN C$-19 TO )
 560 PRINT
 570 NEXT I
 580 CLS
 590 PRINT 'BUENO ';A$
 600 PRINT AT 2,0;'VAMOS A EMPEZAR LA SUMA'
 610 PAUSE 150
 620 CLS
 630 FOR I=1 TO A
 640 PRINT AT I+8-A/2,15-B/2;B$(I,20-B TO )
 650 NEXT I
 660 PRINT TAB 16-B/2;
 670 FOR I=1 TO B
 680 PRINT '-';
 690 NEXT I
 700 LET C=0
 710 LET D=17
 720 FOR I=B TO 2 STEP -1
 730 LET D=15-B/2+I
 740 PRINT  PAPER 6;AT 10+A/2,D;'?'
 750 LET C$=INKEY$
 760 IF C$>='0' AND C$<='9' THEN  GO TO 800
 770 PRINT AT 10+A/2,D;' '
 780 LET C$=INKEY$
 790 IF C$<'0' OR C$>'9' THEN  GO TO 740
 800 PRINT AT 10+A/2,D;C$
 810 GO SUB 1270
 820 LET F=E-10*INT (E/10)
 830 IF VAL C$<>F THEN  GO TO 880
 840 PRINT AT 0,0;'MUY BIEN ';A$;' SIGUE'
 850 PAUSE 100
 860 LET C=INT (E/10)
 870 GO TO 900
 880 PRINT AT 0,0;'NO ';A$;', NO SON ';C$
 890 PAUSE 100
 900 PAUSE 100
 910 PRINT AT 0,0;'
 920 IF VAL C$<>F THEN  GO TO 740
 930 NEXT I
 940 PRINT AT 0,0;'VENGA ';A$;' YA ACABAMOS'
 950 PRINT 'CUAL ES EL ULTIMO NUMERO?'
 960 INPUT '<ENTER>',F
 961 LET K$=INKEY$
 970 GO SUB 1270
 980 IF E=F THEN  GO TO 1010
 990 PRINT AT 0,0;'NO ES ESO. PRUEBA OTRA VEZ'
1000 GO TO 960
1010 LET C$=STR$ F
1020 PRINT AT 10+A/2,D-LEN C$;C$
1030 PRINT AT 0,0;
1040 FOR J=0 TO 56
1050 PRINT ' ';
1060 NEXT J
1070 PRINT AT 0,0;'FANTASTICO ';A$
1080 PRINT AT 2,0;'CUANDO HAYAS COPIADO PULSA ''S'''
1090 LET C$=INKEY$
1100 IF C$<>'S' THEN  GO TO 1090
1110 CLS
1120 PRINT AT 0,0;'PROBAMOS CON OTRA SUMA?'
1130 INPUT C$
1140 IF C$<>'SI' THEN  GO TO 1170
1150 CLS
1160 GO TO 70
1170 IF C$<>'NO' THEN  GO TO 1220
1180 CLS
1190 PRINT AT 9,9-LEN A$/2;'HASTA OTRA ';A$
1200 PRINT AT 11,4;'YA PUEDES DESCONECTARME'
1210 GO TO 10000
1220 PRINT
1230 PRINT 'COMO QUE ''';C$;''''
1240 PRINT
1250 PRINT 'NO ENTIENDO ESO. DIME SI O NO'
1260 GO TO 1130
1270 LET E=C
1280 FOR J=1 TO A
1290 IF B$(J,20-B+I)=' ' THEN  LET B$(J,20-B+I)='0'
1300 LET E=E+VAL B$(J,20-B+I)
1310 NEXT J
1320 RETURN
3000 REM [INV.]SBR.ESCRITURA[TRUE]
3010 FOR F=1 TO LEN Q$
3020 PRINT  PAPER PAP; INK K; FLASH FL;AT FILA,COL+F;
Q$(F)
3025 BEEP .01,INT (RND*10)
3030 NEXT F
3040 RETURN
3050 REM [INV.]SBR.BORRADO ESCRITURA[TRUE]
3060 FOR F=1 TO LEN Q$
3070 PRINT  PAPER 1; FLASH 0;AT FILA,COL+F;' '
3080 NEXT F
3090 RETURN
```

CONTROL DEL CASETE

Ya conocemos por capítulos anteriores los principales comandos para el manejo del casete. Estos comandos son:

SAVE, VERIFY y **LOAD**

De momento, sabemos de su empleo para obtener copia en cinta de un programa en memoria, verificar que el contenido de ésta se ajusta al de la copia en cinta, cosa tan importante como la anterior y, por último, de la posibilidad de recuperar lo grabado después de apagado el ordenador.
En definitiva, tenemos un conocimiento restringido de las posibilidades del casete, al estar limitado al manejo de programas exclusivamente. Sin embargo, sobre este dispositivo podemos almacenar también datos, producidos por nuestros propios programas, tanto en el formato de variables BASIC como de zonas de memoria.
Sobre el primero de estos formatos, podemos decir que nos permite el almacenamiento en casete para su posterior recuperación del contenido de matrices numéricas y alfanuméricas.
Sobre el segundo, el más versátil de ellos, diremos que permite almacenar bloques de memoria de longitud variable. Esto es lo mismo que abrir la posibilidad de grabar en casete, tanto programas en código máquina como contenidos de pantallas, o datos en formato binario.
Además, existe una nueva sentencia que todavía no hemos comentado: **MERGE**. Esta facilita la fusión del programa contenido en la memoria del ordenador con otro cargado desde cinta.
Este sistema de carga es un tanto especial, de forma que las líneas y variables ya existentes en el programa en memoria, ven reemplazados sus



Los nombres de los bloques de datos para grabación deben tener entre uno y diez caracteres.

El casete es el principal medio de almacenamiento masivo de datos en el Spectrum.

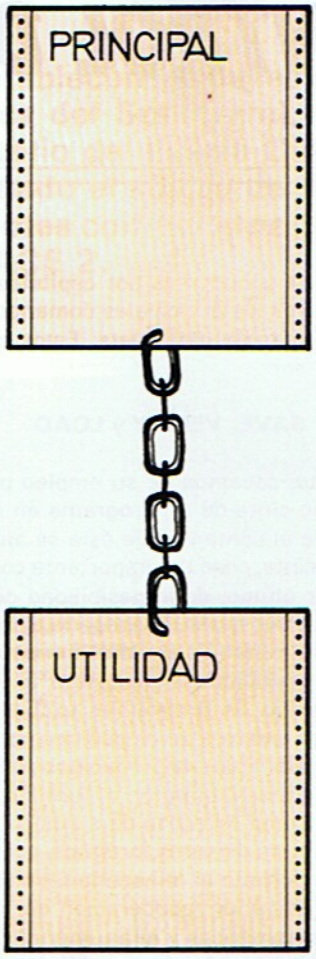
¡!

Una opción de **SAVE** es el "auto **RUN**", que permite que un programa se ejecute de forma automática, a partir de la línea que deseemos, cuando se cargue desde el casete.

*

Las sentencias de manejo del casete nos sirven para obtener copia en cinta del contenido de la memoria, verificar que la copia se ha efectuado correctamente, y recuperar lo grabado después de apagado el ordenador.

contenidos por los nuevos; mientras que las líneas, tanto del programa residente como del almacenado en cinta, sin correspondencia, son incluidas en el programa conjunto sin alteración. Por ejemplo, **MERGE** es especialmente útil para reducir el tiempo de carga de un grupo de programas largos, que hacen uso de un bloque de subrutinas común.
También permite el mantenimiento en memoria de dos o más programas diferentes, siempre que se ajuste la numeración de las líneas, de forma que no se produzcan coincidencias. De este modo, podemos tener un programa principal siempre en memoria, para cargar según nuestro criterio uno determinado de los contenidos en el casete.

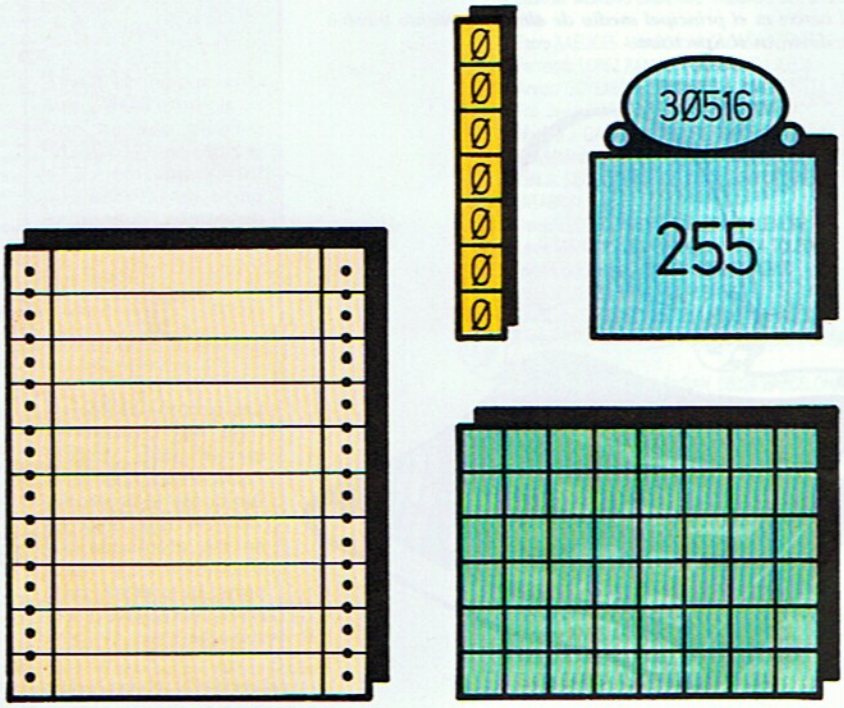


MERGE *permite el encadenamiento de rutinas de utilidad con los programas principales.*

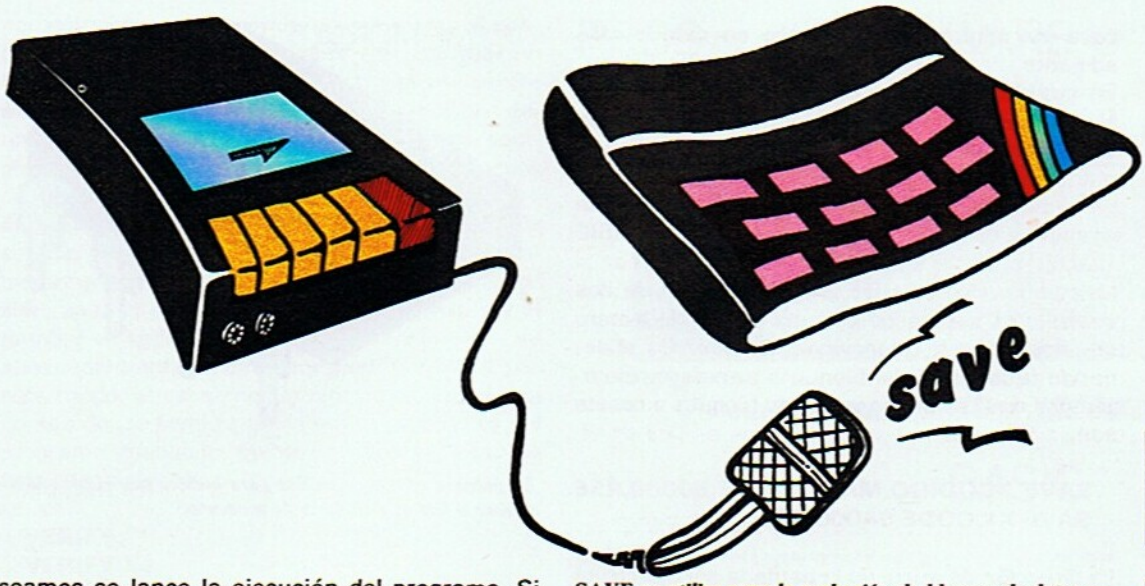
USO EXTENDIDO DE SAVE

La sentencia **SAVE** permite almacenar en casete el contenido de la memoria.

En el casete no sólo se pueden almacenar programas, sino también matrices numéricas, alfanuméricas y zonas de memoria.



Cuando se trata de programas, es obligatorio especificar como parámetro el nombre del mismo, no pudiendo ser una cadena vacía ni superior a 10 caracteres de longitud, puesto que en cualquiera de los dos casos, obtendríamos el mensaje de **C Nonsense in BASIC (C Sin sentido en BASIC).**
Eso sí, como es lógico, puede utilizarse como nombre tanto una expresión entre comillas como una variable de cadena, cumpliendo las condiciones descritas.
Otra opción interesante para el almacenamiento de programas en cinta es el «auto **RUN**". Este sistema permite que un programa se ejecute de forma automática, a partir de la línea que deseemos, la próxima vez que se cargue desde casete, después de efectuado un **SAVE** con estas características.
Para ello, basta con añadir un segundo parámetro compuesto por la palabra BASIC **LINE**, seguida del número de líneas a partir de la cual de-



SAVE se utiliza para la grabación de bloques de datos en cinta.

seamos se lance la ejecución del programa. Si éste debe ejecutarse desde su comienzo, puede escribirse **LINE 0**. En cualquier caso, el nombre de «auto **RUN**» no es exacto, puesto que se simula realmente una instrucción **GO TO** a la línea especificada, respetándose el contenido de las variables.
A continuación veremos algunos ejemplos de uso de **SAVE** con programas:

```
SAVE "PROG"
SAVE X$ LINE 0
SAVE X$
SAVE "PRG001.BAS" LINE 100
```

SAVE puede también emplearse para almacenar matrices en cinta. Para ello, nos basta con especificar el nombre de la matriz a través de un uso especial de la sentencia **DATA**, indicando a continuación la naturaleza de los datos con una pareja de paréntesis:

```
SAVE "AGENDA.DAT" DATA A$()
SAVE X$ DATA A$()
```

Por último, queda una posibilidad más de desarrollo de **SAVE**, consistente en la copia en cinta de bloques de memoria. Dentro de ella, y como un caso especial, estudiaremos la forma de almacenar en casete el contenido de la pantalla. Como ya hemos dicho, la pantalla consta de 768 posiciones, cada una de ellas compuesta por 64 bits (8 bytes), por ello, son necesarios 6912 bytes para almacenar el contenido total de la pantalla, teniendo en cuenta que cada una de estas posiciones lleva aparejado un byte de «atributos»,

La palabra LINE a continuación de SAVE se utiliza para la autoejecución automática de programas.

¡!

Los principales comandos para el manejo del casete son: **SAVE, VERIFY** y **LOAD**.

*

Cuando se usa **SAVE**, es obligatorio especificar un nombre, no pudiendo ser una cadena vacía ni superior a 10 caracteres de longitud.

*

Para utilizar **SAVE** con matrices, basta con especificar la palabra clave **DATA**, seguida del nombre de la matriz y una apertura y cierre de paréntesis.

*

Además de programas, en casete podemos almacenar también datos, tanto en el formato de variables BASIC como en bloques de memoria.

cuya estructura comentaremos en detalle más adelante.
En cualquier caso, la pantalla sigue siendo un bloque, aunque particular dentro de la memoria del ordenador, por ello puede recurrirse igualmente a su almacenamiento en casete, por el procedimiento general destinado a los bloques de memoria, es decir, mediante la palabra BASIC **CODE**.
La palabra clave **CODE** debe ir seguida de dos parámetros separados por una coma; el primero de ellos indica la dirección de comienzo, y el segundo la longitud del bloque a transferir. Ejemplos válidos de transferencia de bloques a casete son:

SAVE "CODIGO.MAQ" CODE 60000,256
SAVE X$ CODE 94000,512

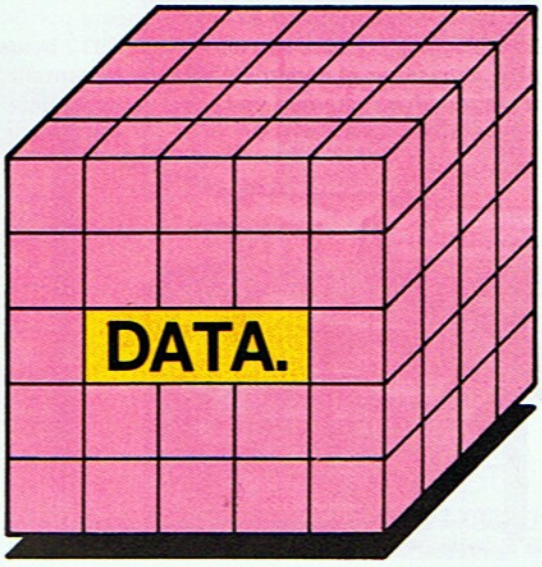
La dirección de inicio de la pantalla del Spectrum es 16384, por lo que para obtener una copia de la pantalla en cinta, podemos escribir:

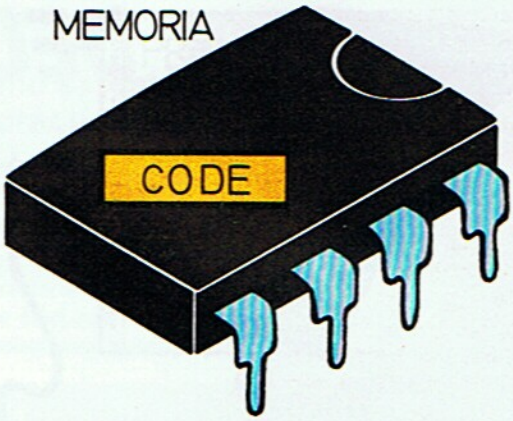
SAVE nombre CODE 16384,6912

No obstante, y debido a la relativa frecuencia de empleo de la copia de pantallas a casete, el Spectrum está dotado de una palabra clave especializada para realizar esta operación, de forma más simple; esta palabra es **SCREEN$**. Con el empleo de **SCREEN$**, el almacenamiento en cinta se reduce a escribir:

SAVE "nombre"SCREEN$

La palabra DATA *a continuación de una sentencia de control del casete, indica que el bloque de datos corresponde a una matriz.*





La palabra CODE *se utiliza para indicar que el bloque a grabar o leer es una zona de memoria.*

Una cosa muy importante debemos recordar sobre **SAVE**. Cuando el programa es almacenado en cinta, lo son al mismo tiempo que las líneas del programa las variables y matrices del mismo, con los contenidos que tienen en el momento de ejecutar esta sentencia, es decir, se efectúa un "vuelco" completo de la memoria BASIC.
Este hecho es muy positivo, si pensamos que en una ejecución posterior del programa nos bastará con cargar éste por medio de **LOAD**, sin preocuparnos de hacer lo propio con los datos correspondientes, pudiendo continuar así con la ejecución, sin más, del programa interrumpido.
En cualquier caso, si no vamos a utilizar esta posibilidad, podemos optar por efectuar un **CLEAR** antes del **SAVE**, o bien un **RUN** después del **LOAD**. Desde luego, la primera opción es la deseable, puesto que ocupa menos espacio en la cinta.

USO EXTENDIDO DE VERIFY

Una vez realizada la operación de copia en cinta, se impone la verificación de que todo ha marchado correctamente; sobre todo antes de apagar el ordenador, dado que perderíamos la información definitivamente.
Para ello, el BASIC dispone de la sentencia **VERIFY**, aplicable a todos los casos explicados anteriormente, con excepción de **SCREEN$**, como veremos más adelante.
Para verificar un programa, es necesario posicio-

¡!

La opción "auto **RUN**" requiere especificar la palabra BASIC **LINE**, seguida del número de línea a partir de la cual deseamos se lance la ejecución del programa.

*

Con el uso de **SCREEN$**, el almacenamiento en cinta de pantallas se reduce a escribir: **SAVE "nombre"SCREEN$**.

*

Al cargar programas con **LOAD**, además de las líneas de programa se lee el área de variables del mismo.

*

La pantalla consta de 768 posiciones, cada una de ellas compuestas por 8 bytes, por ello, son necesarios 6912 bytes en total, teniendo en cuenta que cada una de estas posiciones lleva aparejado un byte de «atributos».

narse previamente al comienzo del mismo. A continuación se plantean dos opciones, efectuar un **VERIFY** del primer programa que se encuentre en la cinta, a partir de la posición en que nos detuvimos; u optar, si no existe la seguridad completa de que el próximo programa sea el adecuado, por especificar el nombre de éste.
El primer sistema es más rápido, el segundo más seguro. No obstante, es de uso generalizado entre programadores el empleo de la primera opción, dada la precaución de grabar de viva voz el nombre de cada programa antes de comenzar su almacenamiento en cinta por medio de **SAVE**; de este modo, al rebobinar la cinta escuchando su contenido, se localiza fácilmente el comienzo del programa adecuado. Ejemplos válidos de uso de esta función son:

```
VERIFY ""        VERIFY "PROG"
VERIFY X$
```

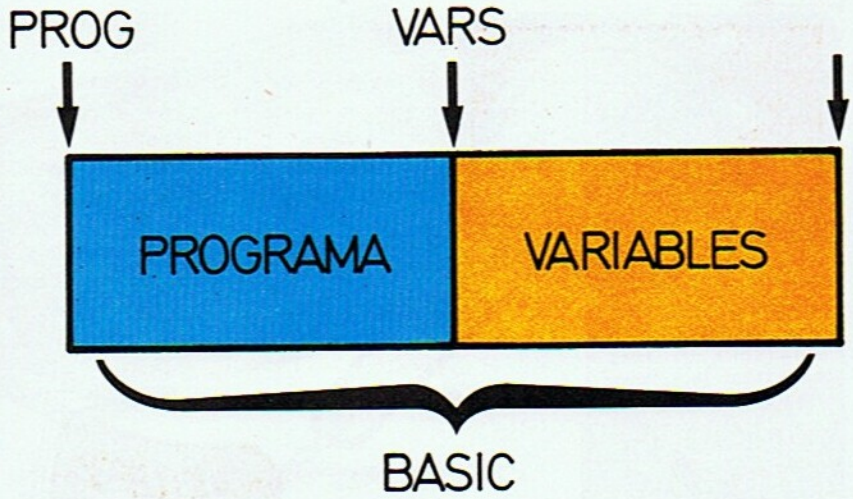
Dado que la sentencia **SAVE** efectúa el vuelco a cinta de la memoria BASIC completa, el **VERIFY** debe ejecutarse seguidamente del **SAVE**, para cerciorarnos de que el programa en memoria y el de cinta son iguales, en lo concerniente al área de variables. Se producirá un error de verificación si alteramos el contenido de alguna de éstas antes del **VERIFY**, hacemos un **RUN** o un **CLEAR**.
VERIFY puede también emplearse para verificar matrices en cinta. En este caso, se debe especificar la palabra clave **DATA** seguida del nombre de la matriz, inmediatamente después de **VERIFY**, para significar la naturaleza de los datos. Ejemplos válidos de uso de esta función son:

```
VERIFY "" DATA A$()
VERIFY "AGENDA.DAT" DATA A$()
VERIFY X$ DATA A$()
```

Existe también la posibilidad de verificar bloques de memoria contenidos en cinta, de forma similar a lo que sucede con **DATA**, a través del uso combinado de **VERIFY** seguido de la palabra **CODE**.
En este caso, está permitido también el empleo de una variable de cadena que contenga el nombre del programa, e incluso hacer caso omiso del nombre, con lo cual se verificaría el primer **CODE** que apareciera en la cinta. Ejemplos válidos de uso de esta función son:

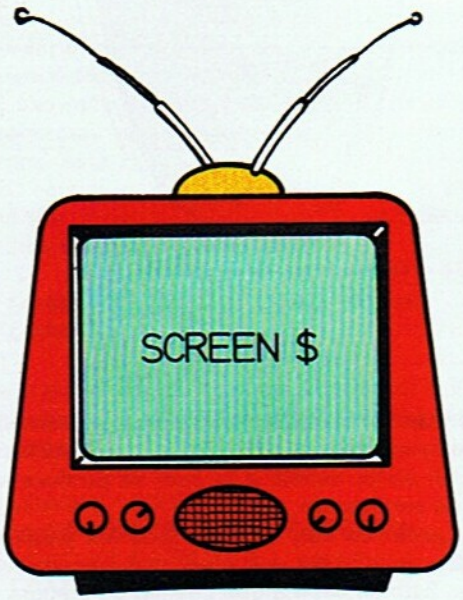
```
VERIFY "" CODE
VERIFY "CODIGO.MAQ" CODE
VERIFY X$ CODE
VERIFY "" CODE 64000
VERIFY "" CODE 64000,512
VERIFY "" CODE A,B
```

Por lo que hemos visto, y esto son sólo algunas de las posibilidades, la función **VERIFY** es de lo más versátil en cuanto a codificación, pues permite especificar de forma opcional la dirección de comienzo del bloque de memoria y su longitud. Esto es posible porque cuando se efectúa el **SAVE** de un **CODE** en cinta, se almacenan al mismo tiempo tanto el nombre como la dirección de comienzo y longitud del bloque.



La memoria BASIC *está constituida por el programa más su área de variables.*

SCREEN$ *se emplea en la grabación y carga de pantallas, y equivale a* **CODE 16384,6912**.

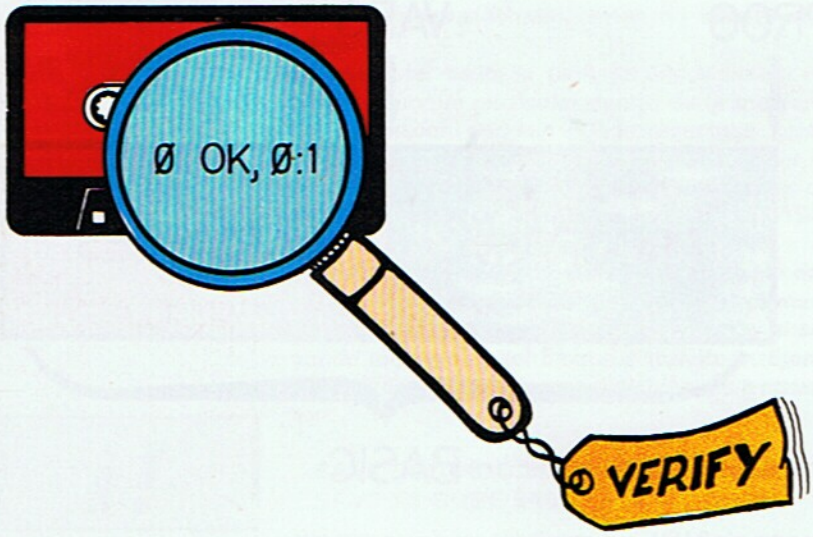


¡!

La dirección de inicio de la pantalla del Spectrum es 16384, por lo que para obtener una copia de la pantalla en cinta, podemos escribir: **SAVE nombre CODE 16384,6912**.

*

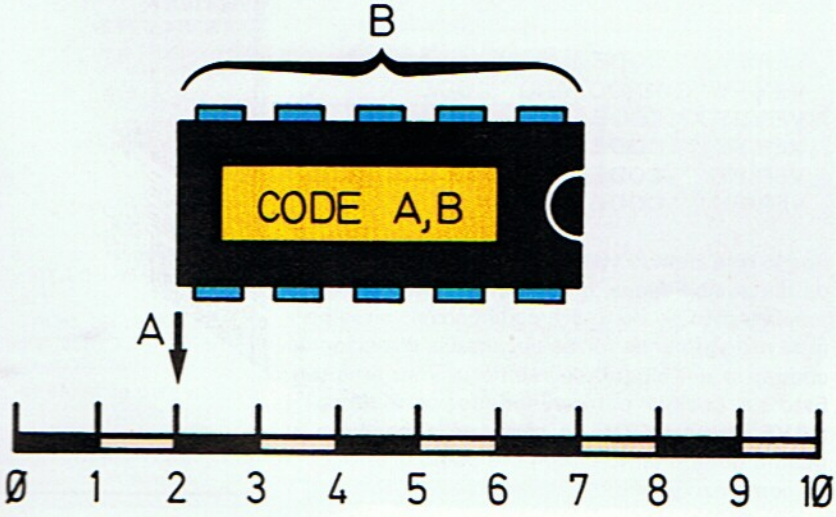
VERIFY sirve también para verificar matrices en cinta, con **DATA** y el nombre de la matriz; o bloques de memoria, con **CODE**, y la dirección de comienzo y número de bytes opcionalmente.



La sentencia VERIFY *se utiliza para comprobar la corrección de las grabaciones.*

De esta forma, cuando nos disponemos a efectuar el **VERIFY**, podemos permitirnos el lujo de omitir algunos de los parámetros, puesto que el ordenador tomará por defecto los almacenados previamente en la cinta, por medio de la última operación **SAVE**.
Otro hecho que puede resultar de interés, es especificar tanto el nombre del **CODE** como su dirección de comienzo y longitud como variables, la primera de ellas de cadena y las dos restantes numéricas.

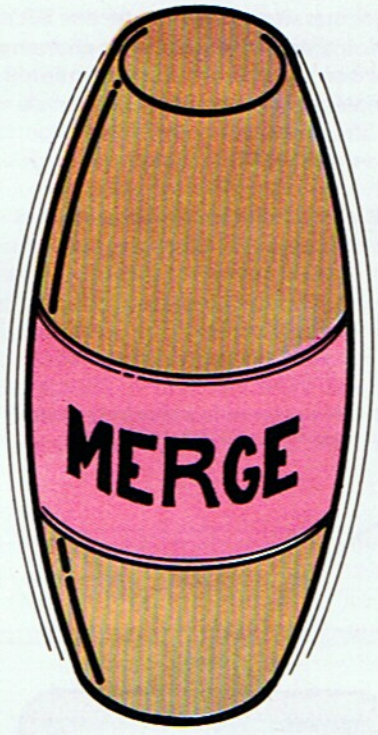
El primer parámetro que sigue a CODE *indica su comienzo, y el segundo su longitud.*



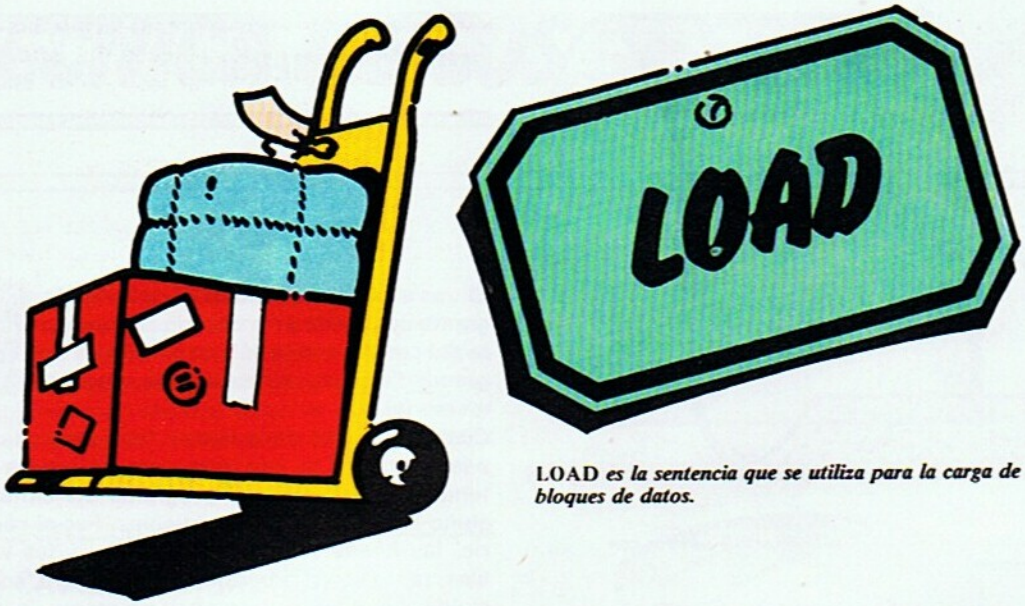
Por este procedimiento, podemos incluir una subrutina de verificación de bloques útil para cualquiera de ellos, con sólo dar los valores adecuados a las variables antes de entrar en ella:

VERIFY X$ CODE A,B: RETURN

Otro uso accesorio de **VERIFY**, puede ser el de verificar sólo una parte del bloque de memoria contenido en cinta, o la de comprobar el bloque en cinta con uno en memoria situado en diferente posición.
Por último, deberíamos comentar la posibilidad de verificar el almacenamiento en cinta de pantalla, aunque desgraciadamente esto no es factible, al menos por el procedimiento normal.
Esto se debe a que al comenzar la verificación y encontrar el ordenador el nombre de la **SCREEN$** en la cinta, imprime el correspondiente mensaje en pantalla desvirtuando su contenido, de forma que ya no coincide con el almacenado en cinta, debido a lo cual se produce un error de verificación.
Como podemos ver, en este caso los palabras **CODE** y **SCREEN$** son absolutamente equivalentes. Esto se debe a que cuando se efectúa el



MERGE *sirve para la mezcla de áreas BASIC.*



LOAD es la sentencia que se utiliza para la carga de bloques de datos.

SAVE de la pantalla con la palabra clave **SCREEN$**, quedan almacenados en cinta los valores de comienzo y longitud del **CODE**, por ello, es indiferente efectuar la verificación posterior con una palabra clave u otra.

USO EXTENDIDO DE LOAD

Después de realizar las operaciones de almacenamiento y verificación en cinta, se puede desconectar el ordenador con la seguridad de que no perderemos los datos, pudiendo recuperarlos cuando deseemos a través de la sentencia **LOAD**. El uso más inmediato de **LOAD** es la carga de programas en memoria. Como ya hemos dicho antes, esta carga se produce del programa BASIC completo, es decir, de las líneas de programa y la correspondiente área de variables. Formatos válidos de **LOAD** son:

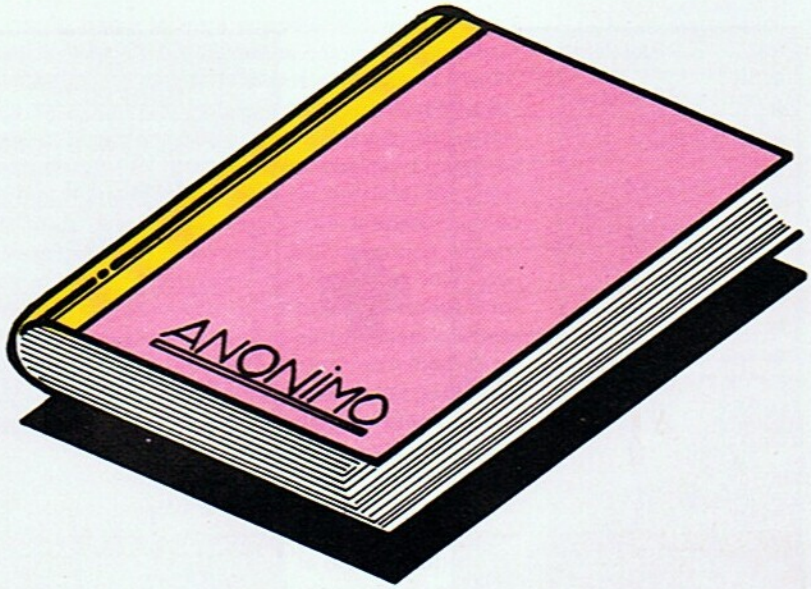
```
LOAD ""          LOAD "PROG"
LOAD X$
```

La segunda opción se corresponde con la lectura de casete de matrices de datos numéricas y alfanuméricas. Son formatos válidos:

```
LOAD "" DATA A$()
LOAD "MATRIZ.DAT" DATA A$()
LOAD X$ DATA A$()
LOAD X$ DATA A()
```

La tercera opción permite la carga de **CODE**. A través de esta opción, pueden cargarse en memoria los bytes procedentes de la cinta en las mismas posiciones en que fueron grabados si no se especifican los parámetros de comienzo y longitud.
Si se suministran los parámetros, puede conseguirse la carga de lo almacenado en cinta en un lugar distinto de la memoria, o cargar un número menor de bytes de los contenidos en el **CODE** que se lee. Algunos ejemplos, en todo similares a **VERIFY** son:

En las operaciones de lectura no es preciso especificar nombre.

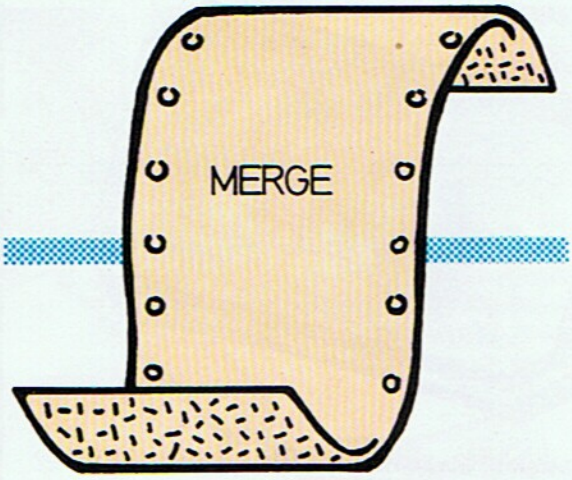


¡!

Una posibilidad más de **SAVE**, es la copia en cinta de bloques de memoria, y dentro de ella, como caso especial, el almacenamiento en casete del contenido de la pantalla.

*

La palabra BASIC **CODE** debe ir seguida de dos parámetros separados por una coma; el primero de ellos indica la dirección de comienzo, y el segundo la longitud del bloque a transferir.



La instrucción MERGE *sólo se puede utilizar con programas.*

```
LOAD ""
LOAD "CODIGO.MAQ" CODE
LOAS X$ CODE
LOAD X$ CODE A,B
```

Por último, podemos cargar las pantallas escribiendo:

```
LOAD ""
LOAD "" SCREEN$
LOAD"nombre"
LOAD"nombre"SCREEN$
```

EMPLEO DE MERGE

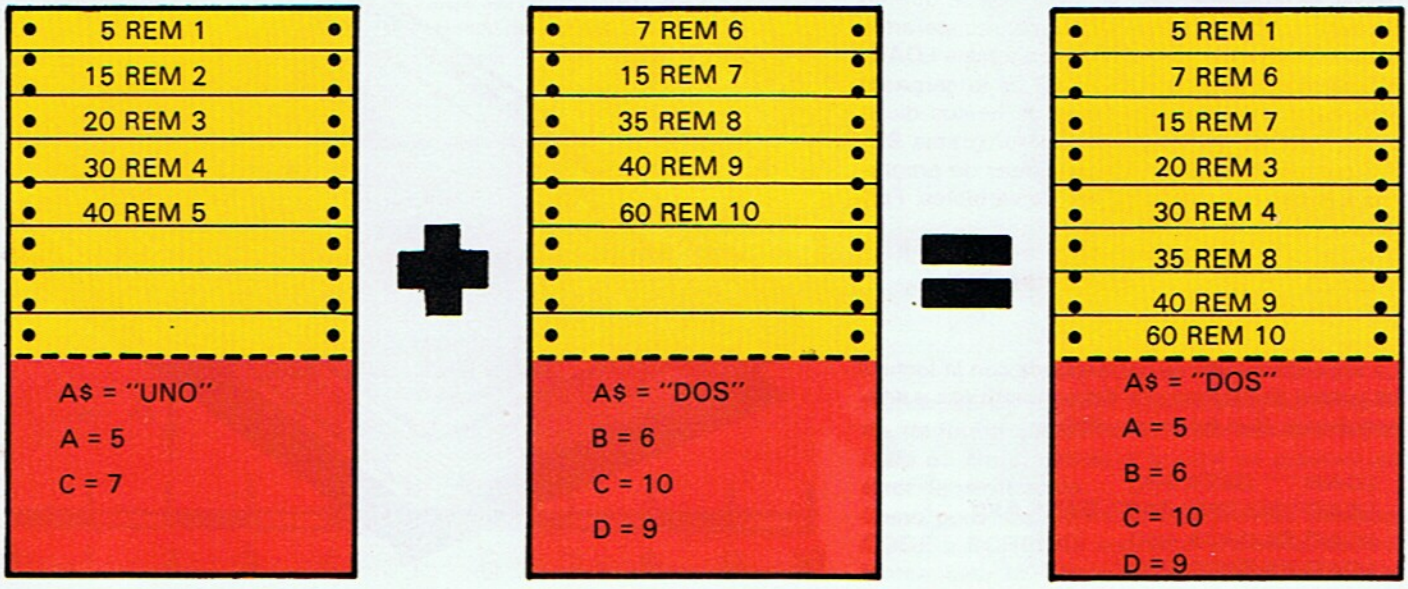
El uso exclusivo de **MERGE** es la fusión del programa contenido en memoria con uno proviniente del casete, con la sóla condición de que el programa "suma", no exceda la capacidad de la memoria.

Como ya hemos dicho antes, los números de líneas comunes se ven reemplazados por los contenidos que llegan del programa en cinta, así como los valores de las variables. Por el contrario, las líneas sin correspondencia, y las variables, tanto de un programa como de otro, son incluidas sin modificación en el programa "suma".

MERGE no está permitido en combinación con las palabras clave **DATA** ni **CODE**. Si se utiliza con programas grabados con la opción de "auto **RUN**", les priva de esta característica. Ejemplos de uso correcto de **MERGE** son:

```
MERGE ""      MERGE "PROG"
MERGE X$
```

Proceso de unión de bloques mediante MERGE.

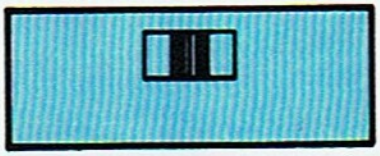


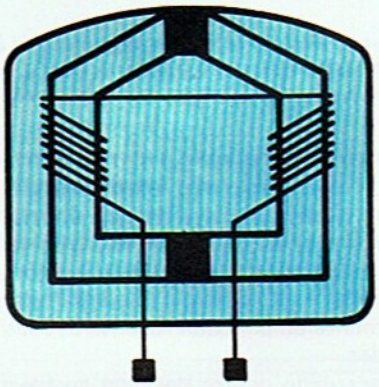
TECNICA DE LOS CASETES

EN el capítulo anterior, estudiamos los casetes desde un punto de vista general, refiriéndonos a ellos según sus características. Hoy acometemos un objetivo más técnico, tanto en lo referente al aparato grabador/reproductor, como a la cinta casete en sí.

LAS CABEZAS MAGNETICAS

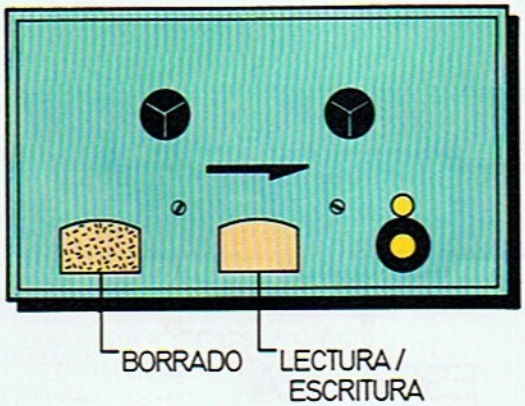
No entraremos en detalle sobre toda la circuitería interna que conforma un casete de audio convencional. Sin embargo, vamos a comentar el componente principal de los sistemas basados en cinta magnética: los cabezales.
Estos son de tres tipos: grabación o escritura, reproducción o lectura, y borrado. Los equipos modernos utilizan un único cabezal para las labores de grabación y lectura. Podemos observarlos con detalle si mantenemos abierta la tapa de nuestro casete, y pulsamos PLAY. Los cabezales se desplazarán hacia la zona ocupada por la cinta, que como es lógico no habremos colocado en su alojamiento para evitar que nos dificulte la visión. Uno de ellos está envuelto en una pequeña cápsula de color metálico: es la cabeza de lectura/escritura. El otro, de color blanco o grisáceo normalmente, es la cabeza de borrado. En algunas grabadoras esta cabeza solo aparece cuando hemos pulsado PLAY+RECORD, es decir, mientras estamos efectuando un trabajo de escritura, pues su misión consiste en borrar la antigua información que contuviera la cinta, dejándola preparada para que la cabeza de escritura ejecute su trabajo sin posibilidad de interferencias parásitas. Los cabezales son, en esencia, un electroimán, es decir, el conjunto formado por un núcleo de material magnético sobre el cual se encuentra arrollada una bobina eléctrica. A través de ella se hace circular la corriente que genera el campo magnético. Este provoca la orientación de las partículas que recubren la cinta de casete en





Los cabezales son esencialmente electroimanes; un núcleo de material magnético sobre el cual se encuentra arrollada una bobina eléctrica.

En los casetes podemos distinguir una cabeza de borrado y otra de lectura/escritura.



¡!

Los cabezales de lectura y escritura se encuentran envueltos en una misma cápsula metálica, y se sitúan hacia la mitad del mecanismo que aparece al pulsar PLAY.

*

La cabeza de borrado se sitúa antes de la de escritura, y tiene por fin borrar la cinta antes de su llegada al punto de grabación.

La cabeza de borrado se encarga de preparar la cinta para la posterior escritura.

unas u otras direcciones determinadas, almacenando de esta manera la información.
Durante el proceso inverso, es decir, mientras efectuamos operaciones de lectura, conforme la cinta va pasando por delante de la cabeza de reproducción, ésta le induce una señal, que convenientemente amplificada reconstruye los sonidos originales.
El encapsulado dentro del cual se haya alojado el electroimán es fundamental. Sirve como blindaje para evitar posibles interferencias por parte de campos magnéticos exteriores. De otra manera, la información podría tergiversarse o quedar deteriorada. Por tanto, es importante no manipular los cabezales con objetos punzantes, tales como un destornillador, por ejemplo, que puedan llegar a deteriorarlo.

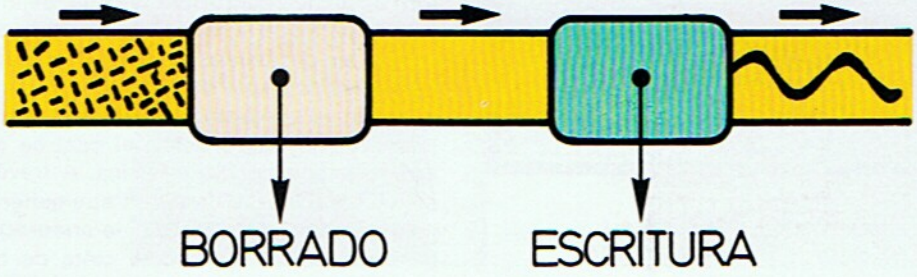
El encapsulado dentro del cual se encuentra el electroimán es fundamental, puesto que le protege de los campos magnéticos externos.

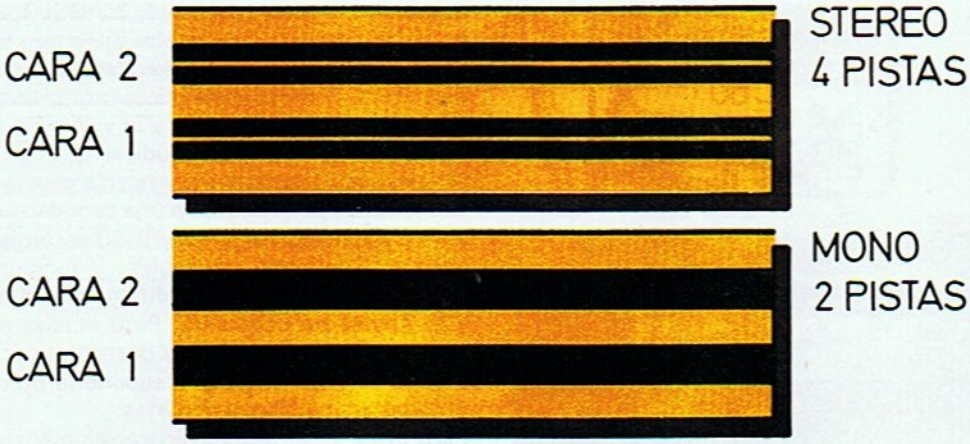
CINTAS

El objetivo de los fabricantes: popularizar la cinta magnética, dotando a los usuarios de un sistema de pequeño tamaño que conviertiera los enormes magnetófonos en mecanismos más reducidos y, por tanto, más asequibles al consumidor; se consiguió plenamente con la introducción de la cinta casete.
Quizá convendría hacer una aclaración sobre los términos casete y grabadora. Casete propiamente dicho, es el conjunto formado por la cinta magnética, más la carcasa plástica que recubre al sistema. La grabadora, desde luego, es el aparato encargado de introducir o recuperar la información (música, voz o datos de un ordenador) almacenada en la cinta magnética.
Muchas veces, se aplica el término casete como sinónimo de grabadora, en el sentido de que nos estamos refiriendo a un sistema de grabación/reproducción, en el cual la cinta magnética se encuentra encapsulada en el interior de una cubierta plástica de dimensiones normalizadas.
Seguramente, nadie fuera capaz al principio de imaginar que el desarrollo de la microinformática, convertiría a los casetes en los soportes más utilizados como medio de almacenamiento masivo de la información.

CONSTITUCION FISICA

Las cintas están formadas por una delgada película plástica, normalmente de poliester, sobre la





La diferencia entre la grabación en modo mono o estéreo afecta al número de pistas contenidas en cada cara.

cual se encuentra depositada una fina capa de materiales ferromagnéticos. Tanto el principio como el final del carrete, son de plástico inerte, sin emulsión magnética, siendo estas zonas más resistentes que el resto de la cinta a las tracciones bruscas que pueden producirse al poner en marcha el aparato.

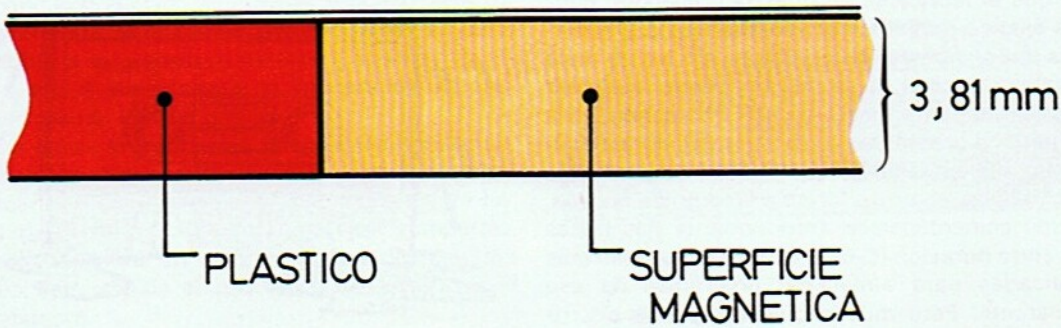
El ancho está normalizado a 3,81 milímetros. La longitud es variable, dependiendo del tipo de cinta escogido, aunque en el caso de los casetes no se habla de longitud, sino de tiempo de duración. Resulta evidente que mayor duración implica mayor longitud, y debido a ello, los espesores de las cintas serán diferentes, según la duración de estas.

Así, podemos encontrar en el mercado cintas especiales para ordenador, de cinco, diez o quince minutos, además de las habituales de cuarenta y cinco, sesenta, noventa o ciento veinte; éstas últimas prácticamente en desuso. La duración queda claramente especificada en la envoltura de la caja que contiene al casete. Normalmente, se emplea el prefijo C, seguido del número de minutos disponibles para grabación, bien entendido que si la cinta es, por ejemplo, una C-60, dispondremos de 30 minutos útiles por cada cara. El calificativo «especial», no quiere decir, que las de corta duración sean las únicas que se utilizan para almacenar la información proviniente de un ordenador, sino que, al ser más cortas, el tiempo de acceso a un programa es menor, factor a considerar si éste es de uso cotidiano.

ESCOGE LA ADECUADA

El material empleado como recubrimiento para transformar la cinta plástica en magnética, nor-

Las cintas están compuestas por una delgada película plástica, recubierta de materiales ferromagnéticos.



¡!

El encapsulado dentro del cual se encuentra alojado el electroimán, sirve para protegerlo de los campos magnéticos externos.

*

La humedad y el calor son los grandes enemigos de las cintas.

*

Los cabezales son, en esencia, un electroimán: es decir, un núcleo de material magnético sobre el cual se encuentra arrollada un bobina eléctrica.

Adquiramos cintas de calidad; nuestro Spectrum notará la diferencia.

malmente es óxido de hierro. En la actualidad, se utilizan otros materiales, como mezclas de óxidos de hierro y cobalto, dióxido de cromo o, en lo que se denominan cintas de metal, una fina película de cobalto metalizado.

Si bien es cierto que éstas últimas ofrecen mejores prestaciones para conservar el último éxito de nuestro cantante favorito, almacenar la «voz» de nuestro Spectrum, no requiere tanta sofisticación. Además, si la grabadora no dispone de conmutadores especiales para trabajar con este tipo de cintas especiales, el gasto habrá sido tan inútil como peligroso, puesto que al ser estos materiales más abrasivos que el óxido de hierro, harán disminuir la vida de los cabezales.

Una forma de reconocer si se trata de una cinta convencional, de las utilizadas por las pequeñas grabadoras portátiles, consiste en observar entre las características el apartado que indica el tipo de polarización o «bias». Deberá aparecer la indicación «Normal».

Pero siempre debemos escoger, cuando menos, cintas de reconocida calidad. Por lo general, es un buen criterio seleccionarlas de la misma marca que el fabricante de nuestra grabadora, pues los ajustes realizados al aparato fueron efectuados tomando éstas como patrón. En las de mala calidad, el nivel de grabación puede disminuir alarmantemente, hasta límites que conducen a la pérdida prematura de la información almacenada; ello es debido a que la capa de partículas magnéticas no es uniforme a lo largo de la cinta.

Como comentábamos anteriormente, las cintas de corta duración (C-5, C-10 o C-15) son las más indicadas para almacenar programas de uso constante. Pero mucho cuidado: puede ocurrir que llevados por un exceso de confianza, la cinta llegue al final antes de que el Spectrum termine de trasmitirle toda la información. Ten en cuenta que una C-5 no es capaz de almacenar más de 12 ó 14 Kb. por cada cara, a la velocidad normal de transmisión de 1500 baudios.

Si buscamos disponer en una sola casete de varios programas que ocupen una cantidad apreciable de memoria, las C-45 y C-60 serán las adecuadas. Son bastante fiables, y tengamos en cuenta que una C-60 puede conservar más de 300 Kbytes en cada cara. Pero mucho cuidado con la calidad: es preferible gastar un poco más, a sufrir la impotencia que supone la pérdida de un programa o bloque de datos.

Para evitar desgracias irreparables o sustos prematuros, conviene, como norma, realizar copias de seguridad de todos los programas. Las cintas de 90 minutos cumplen a la perfección el cometido como archivos maestros de datos y programas, siempre y cuando las conservemos adecuadamente y no las deterioremos por el uso continuado.

En cuanto a las de 120 minutos, aunque su capacidad es mayor, es mejor rechazarlas para su utilización con ordenadores. Salvo honrosas excepciones, y hablando en el más puro sentido infomático, son de «mírame y no me toques». La razón viene justificada en su espesor, que es más fino debido a su longitud.

Los contínuos tirones que provocamos cuando intentamos encontrar un determinado programa, pueden deformarla. En una cinta grabada con música, el efecto será una pequeña distorsión de la melodía, apenas apreciable; pero si contiene la serie de bits que configuran un bloque de infor-

La humedad y el calor son enemigos irreconciliables de las casetes.

¡!

Las cintas están formadas por una delgada película plástica, recubierta de una fina capa de materiales ferromagnéticos.

*

El prefijo C seguido de un número, indica la duración de la cinta por las dos caras.

*

Es necesario alejar las cintas de cualquier campo magnético (televisores, monitores de vídeo, etc...).

mación, nuestro Spectrum es posible que no sea capaz de interpretarlo, y mostrará su disconformidad con un, casi siempre irritante, **R Tape loading error** (Error de carga en cinta).

CONSERVACION DE CINTAS

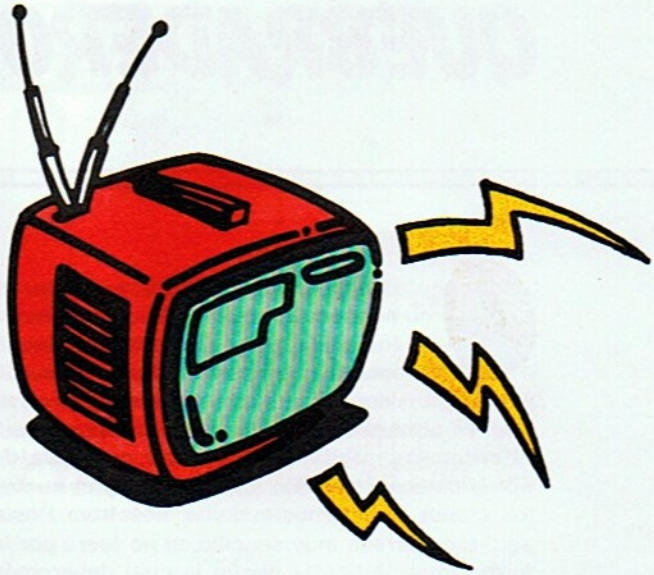
Es relativamente frecuente comprobar, no sin estupor, que al intentar cargar un programa que ayer funcionaba perfectamente, hoy se resiste a entrar. Una de las posibles causas quedó apuntada anteriormente, cuando comentamos las tensiones que sufre una cinta de larga duración con las contínuas paradas y puestas en marcha de la grabadora al buscar un programa. Por ello, el consejo de los fabricantes: «deje pasar la cinta hasta el final, antes de rebobinar», no debe echarse en saco roto. Y tienen razón, pues como ya sabemos, las zonas de principio y fin son más resistentes que el resto de la cinta.

La gran difusión alcanzada por las casetes, se justifica en gran medida por la protección que frente al polvo y manipulaciones externas proporciona la carcasa plástica que recubre a la cinta. Por ello, su vida es, generalmente, superior a la de los rollos de bobina, usados en los magnetófonos, siempre que no nos declaremos «manipuladores oficiales», y sepamos mantener los dedos suficientemente alejados de la película magnética.

La humedad y el calor son enemigos irreconciliables de las casetes. La primera, porque puede llegar a alterar la composición de los materiales ferromagnéticos; el segundo, deformará la carcasa, y el ajuste que por las ventanas de esta deben realizar los cabezales y la cinta, no será lo suficientemente preciso para que la lectura o grabación sean correctas.

Es importante mantener distantes las cintas de cualquier campo magnético. De la misma manera que la grabación se efectúa al actuar éste sobre la cinta, si el campo proviene de otra fuente distinta a la grabadora (televisor, monitor de vídeo, etc...), puede alterar la información almacenada, dando lugar a errores.

Para evitar la posibilidad del borrado accidental, las cintas vírgenes llevan en la parte trasera de la carcasa unas lenguetas destinadas a este fin. Basta forzarlas, ligeramente, para romperlas, quedando libre un orificio donde encajara una guía, que impide el uso de la tecla REC en la grabadora.

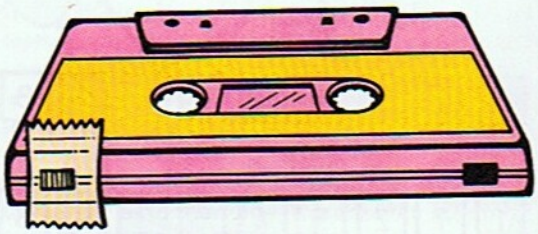


Mantengamos las cintas fuera de cualquier campo magnético, como por ejemplo, los televisores.

Normalmente, cuando adquirimos un programa comercial en cinta, está protegido por este sistema contra el borrado. Pero en cualquier momento podemos recubrir los orificios con cinta adhesiva, u obturarlos con un poco de papel, quedando la casete de nuevo lista para ser utilizada en procesos de grabación.

Como medida de precaución, es conveniente cada vez que terminemos de utilizar una casete devolverla a su estuche, donde permanecerá protegida hasta la siguiente ocasión que la necesitemos, y de no ser posible ésto, rebobinarla completamente, de forma que la zona expuesta al exterior sea el comienzo plástico que no porta información.

Las protecciones contra escritura pueden ser fácilmente evitadas mediante una simple tira de cinta adhesiva.



¡!

Los cabezales son de tres tipos: grabación o escritura, reproducción o lectura, y borrado.

*

Para la grabación de datos no es recomendable el uso de cintas de metal. En el apartado de especificaciones, la característica bias (polarización) deberá ser NORMAL.

QUEBRADERO

QUIZAS el nombre de este juego no nos resulte muy familiar en primera instancia, pero sin duda, en cuanto intentemos resolver el problema que plantea, nos daremos cuenta del por qué de su título: es un auténtico quebradero de cabeza.

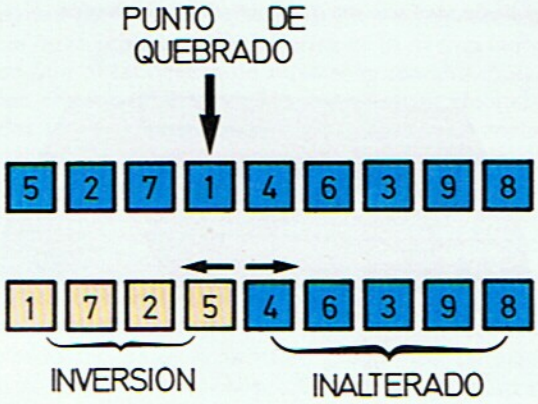
El ordenador propondrá una serie ordenada de números comprendidos entre el uno y el nueve, los cuales deberemos ordenar nosotros. Hasta aquí todo parece muy sencillo, si no fuera por la forma muy particular según la cual deberemos ordenar la serie.

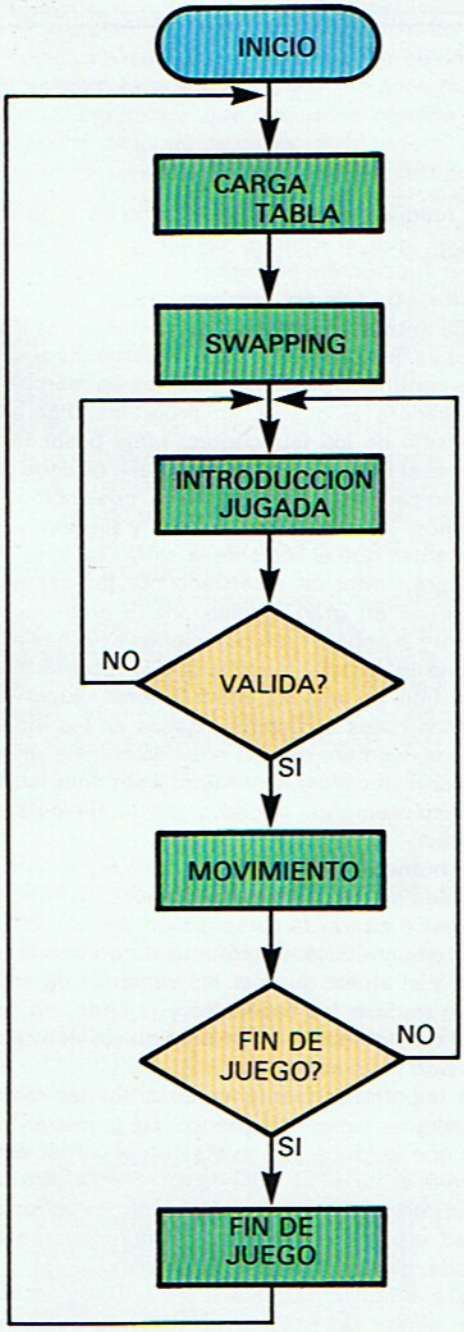
Tal como se presenta en la pantalla, la serie es una fila compuesta por nueve posiciones, las cuales podríamos imaginariamente numerar del uno al nueve, contando de izquierda a derecha. Para resolver el problema, nosotros deberemos ir indicando al ordenador los puntos por los cuales deseamos «quebrar» las diferentes configuraciones que se van formando por quebrado de las anteriores

La nueva serie compuesta será la resultante de invertir las posiciones de los caracteres comprendidos desde la primera posición (elemento de la izquierda) hasta la indicada como punto de quebrado. Así por ejemplo, si la serie fuera...

5 2 7 1 4 6 3 9 8

Los elementos comprendidos desde el primero hasta el punto de quebrado son invertidos, mientras que el resto de la serie permanece inalterada.



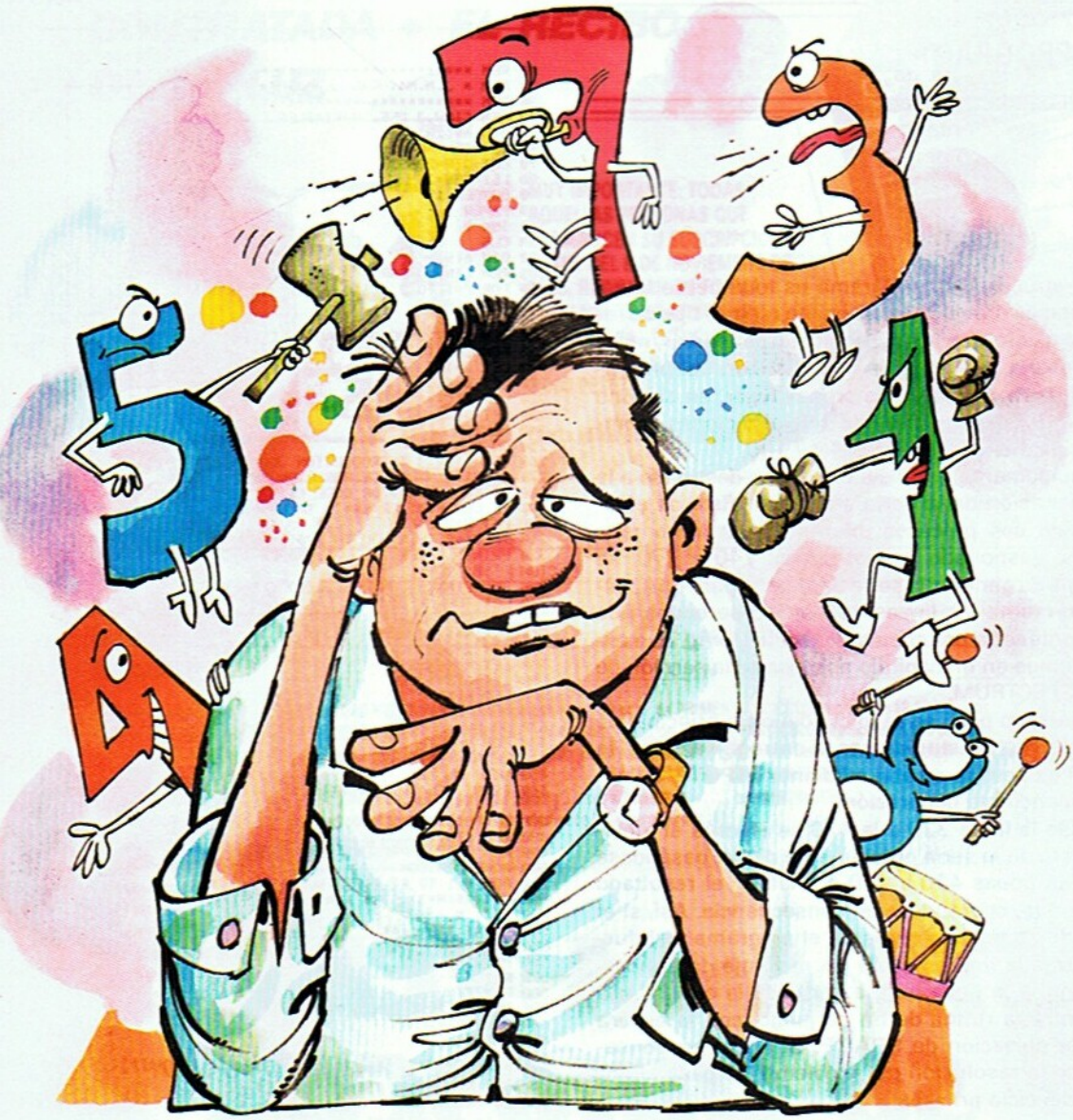


En el diagrama de flujo del programa se puede observar la extremada sencillez del mismo.

y seleccionáramos el cuatro como punto de ruptura, la nueva fila resultante sería...

1 7 2 5 4 6 3 9 8

Como podemos observar, el resto de la serie, comprendido desde el punto siguiente al de ruptura hasta el final de la misma (posición de la derecha) permanece inalterado.
Basándonos en esta única norma, deberemos intentar efectuar los movimientos adecuados para resolver el problema en el menor número posible de intentos.
Cuando hemos conseguido nuestro objetivo, el programa se detiene con un mensaje de enhorabuena, esperando la pulsación de la tecla **SPACE**, o la barra espaciadora en el Plus, para recomenzar una nueva partida, partiendo de otra serie desordenada aleatoriamente.

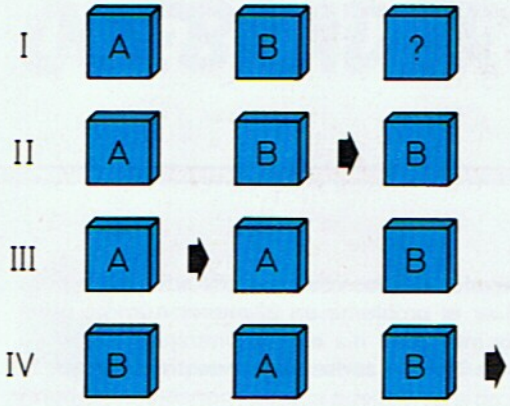


¡!

Las palabras **INV.** y **TRUE** subrayadas y entre corchetes, deben ser introducidas como caracteres **INVERSE VIDEO (CAPS SHIFT + 4)** y **TRUE VIDEO (CAPS SHIFT + 3)**, respectivamente.

*

En esta ocasión, el programa no incorpora gráficos de ningún tipo: ni predefinidos ni de usuario. A ello se debe la ausencia de caracteres simples subrayados.



El sistema de shuffling *empleado sobre la serie es el de* swapping *o intercambio de elementos.*

EL PROGRAMA

La estructura del programa es totalmente lineal y sencilla. Puesto que no existe ningún tipo de record ni información especial que conservar, no es necesaria una zona de inicialización del programa y otra de juego. En primer lugar, se dispone de una zona general de inicializaciones, comprendida entre las líneas 10 y 130.

Seguidamente, se inicia un bloque destinado a la preparación de la serie a ordenar. Esta se efectúa en dos procesos diferentes: el primero de ellos, gestionado por las líneas 140 a 170, se ocupa de generar la serie ordenada, para que posteriormente las líneas 180 a 250 se ocupen de desordenarla por el método de *swapping,* que estudiamos en un capítulo anterior de la sección de TU SPECTRUM.

El próximo paso es el inicio del ciclo principal de programa, constituido por la entrega de datos, la cual es llevada a cabo mediante **INPUT** y su correspondiente depuración.

Desde la línea 330 a la 420 se efectúa el movimiento de la serie propiamente dicho, pasándose en las líneas 430 a 470 a analizar el resultado obtenido, operándose en consecuencia. Así, si el problema ha sido resuelto, el programa se bifurca, tras la impresión de un mensaje de enhorabuena, a la línea 590, a partir de la cual se encuentra la rutina de fin de juego, con la espera de la pulsación de **SPACE**. Si no se ha conseguido la resolución del problema, se pasa al inicio del ciclo principal.

Ya sólo resta comentar la subrutina situada entre las líneas 480 y 570, cuyo fin es la representación en pantalla de cada una de las series obtenidas, evitando la emisión del mensaje de *scroll* mediante el control de la Variable del Sistema SCRCT.

El programa no presenta ninguna dificultad especial para la introducción, puesto que no utiliza gráficos definidos ni implementa código máquina. Sólo se han utilizado los caracteres **INVERSE VIDEO (CAPS SHIFT + 4)** y **TRUE VIDEO (CAPS SHIFT + 3)**, con el fin de resaltar las líneas **REM** que separan los bloques principales de programa; para su representación en el listado se han empleado, respectivamente, las palabras **INV.** y **TRUE** subrayadas y entre corchetes. La grabación se puede conseguir por el método habitual con **SAVE "QUEBRADERO"**, o bien, **SAVE "QUEBRADERO" LINE 10** para la opción de autoejecución.

¡!

Para evitar la aparición del mensaje de *scroll,* se controla el valor que toma la Variable del Sistema SCRCT, situada en la dirección decimal 23692.

*

Para la grabación del programa podemos emplear una instrucción **SAVE "QUEBRADERO"**, o bien, si optamos por el modo de autoejecución: **SAVE "QUEBRADERO" LINE 10.**

```
  10 REM ************************
  20 REM * J.M.MAYORAL SERRANO *
  30 REM ************************
  40 REM [INV.] DEF. VARIABLES [TRUE]
  50 DIM A$(28)
  60 LET CONT=0
  70 LET DIR=23692
  80 LET CON=255
  90 BORDER 1
 100 PAPER 1
 110 INK 9
 120 CLS
 130 DIM A(9)
 140 REM [INV.] CARGA TABLA [TRUE]
 150 FOR P=1 TO 9
 160 LET A(P)=P
 170 NEXT P
 180 REM [INV.] DESORDENACION [TRUE]
 190 FOR C=9 TO 2 STEP -1
 200 LET Q=INT (9*RND)+1
 210 LET W=A(C)
 220 LET A(C)=A(Q)
 230 LET A(Q)=W
 240 NEXT C
 250 GO SUB 480
 260 REM [INV.] INTRODUCCION JUGADAS [TRUE]
 270 INPUT "INTRODUCE POSICION TOPE    ";K$
 280 IF CODE K$<49 OR CODE K$>57 THEN  GO TO 270
 290 IF LEN K$>1 OR K$="" THEN  GO TO 270
 300 LET H=VAL K$
 310 IF H<=9 THEN  GO TO 320
 320 PRINT  PAPER 1;"    "; PAPER 6;" QUEBRADO EN POSI
CION  ";H;" "
 330 REM [INV.] CONTROL  JUEGO [TRUE]
 340 LET TOPE=H/2
 350 LET CONT=CONT+1
 360 FOR N=1 TO TOPE
 370 LET Z=A(N)
 380 LET X=H-N+1
 390 LET A(N)=A(X)
 400 LET A(X)=Z
 410 NEXT N
 420 GO SUB 480
 430 FOR N=1 TO 9
 440 IF A(N)<>N THEN  GO TO 270
 450 NEXT N
 460 PRINT  FLASH 1; PAPER 2;;"          LO CONSEGUIST
E
 470 GO TO 590
 480 REM [INV.]          SCROLL        [TRUE]
 490 REM [INV.] PRESENTACION JUGADA [TRUE]
 500 POKE DIR,CON
 510 FOR N=1 TO 9
 520 LET A$(N*3)=STR$ A(N)
 530 NEXT N
 540 PRINT TAB 1;A$
 550 PRINT
 560 PRINT
 570 RETURN
 580 REM [INV.] FIN DEL JUEGO [TRUE]
 590 DIM A$(28): DIM A(9)
 600 POKE DIR,CON
 610 PRINT
 620 PRINT
 630 PRINT "PULSA <SPACE> PARA OTRA PARTIDA!"
 640 IF INKEY$=" " THEN  RUN
 650 GO TO 640
```

PUNTO A PUNTO

Ya hemos dicho, al hablar de la configuración general de la pantalla del Spectrum, que ésta consta de 24 filas de 32 columnas. A su vez, la pantalla está dividida en dos zonas que podemos denominar como de «impresión» y de «Sistema». La zona reservada a uso del Sistema está compuesta por las dos líneas inferiores de la pantalla, la 22 y la 23, y se emplea para las entradas de datos por medio de **INPUT**, mostrar los mensajes de aviso y error, etc...

La zona de impresión es la accesible directamente por el usuario, y permite la escritura por medio de **PRINT** con sus diferentes calificativos, **TAB**, **AT**, apóstrofe ('), coma (,) y punto y coma (;). Cada una de las posiciones de impresión de las 22 líneas accesibles tiene una configuración de ocho filas de ocho puntos, en total 64, que componen la configuración de cada carácter, de forma similar a como vimos en la definición de caracteres por el usuario.

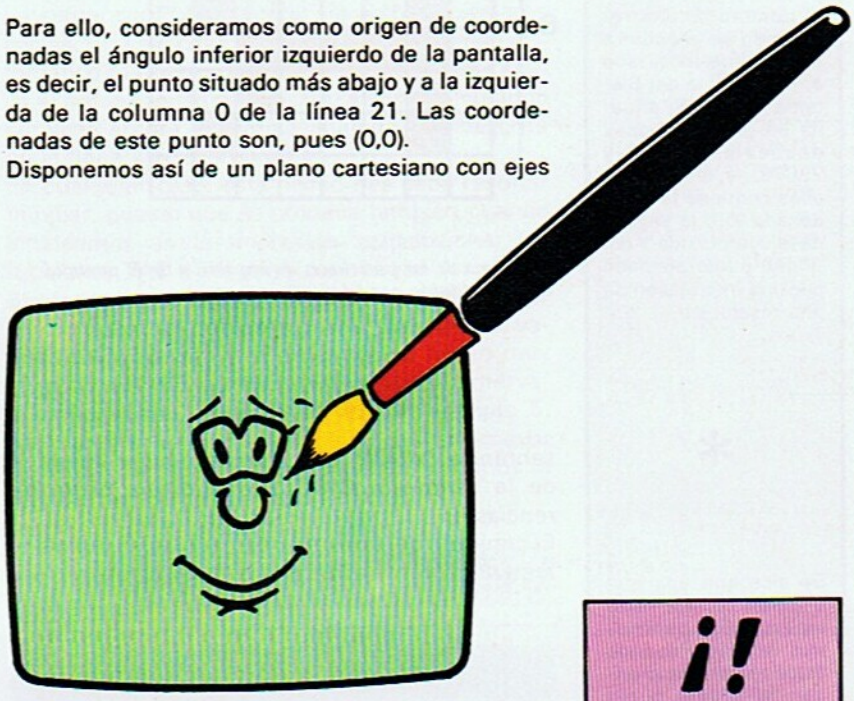
Cada uno de esos puntos elementales de que se compone la imagen en pantalla, recibe el nombre de *pixel*, abreviatura de las palabras inglesas *picture element*, cuyo significado es «elemento de imagen».

Sabido esto, estamos en condiciones de pensar en la pantalla como una malla de 256 × 176 (45056 puntos en total). El Spectrum cuenta con un modo de alta resolución, que permite encender o apagar cada uno de esos puntos con sólo suministrar sus coordenadas.

Para ello, consideramos como origen de coordenadas el ángulo inferior izquierdo de la pantalla, es decir, el punto situado más abajo y a la izquierda de la columna 0 de la línea 21. Las coordenadas de este punto son, pues (0,0).

Disponemos así de un plano cartesiano con ejes horizontal y vertical. Por seguir la notación convencional, nos referiremos al eje horizontal como EJE X y al vertical como EJE Y.

La acción de encender y apagar puntos concretos de la pantalla, se lleva a cabo a través de la sentencia **PLOT**. Además, el Spectrum dispone de la



En el Spectrum disponemos de unas sentencias de gran precisión que nos permiten manejar la pantalla punto a punto.

La zona reservada al Sistema está compuesta por las dos líneas inferiores de la pantalla (líneas 22 y 23).

¡!

La pantalla está dividida en 24 filas de 32 columnas, numeradas a partir de cero, de las cuales las dos últimas filas (22 y 23) están reservadas al Sistema.

*

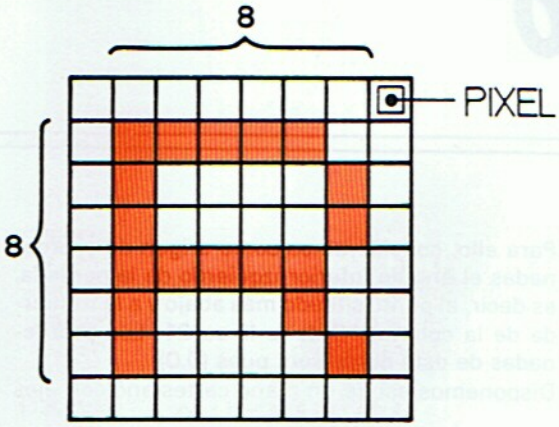
Cada uno de los puntos elementales que componen la pantalla se denomina *pixel*, abreviatura de las palabras inglesas *picture element*, cuyo significado es «elemento de imagen».

¡!

El puntero de alta resolución se encuentra físicamente localizado en la Variable del Sistema COORDS, situada en las direcciones decimales 23677 y 23678; la primera de ellas contiene la coordenada X, y la segunda la coordenada Y del último punto afectado por una instrucción de alta resolución.

*

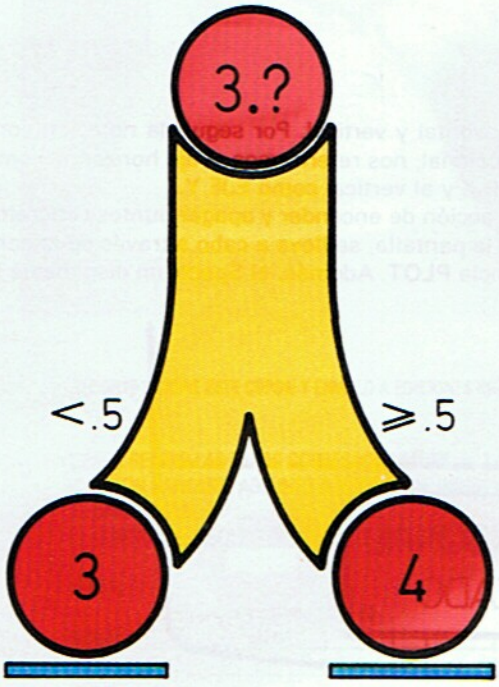
Se dice que una instrucción de alta resolución utiliza parámetros relativos, cuando éstos toman como origen el último punto afectado por una instrucción de este género, es decir, el puntero de alta resolución. Las sentencias de alta resolución del tipo relativo son **DRAW**, en su formato de línea, y en el de arco circular. Por tanto, sus parámetros pueden adoptar tanto valores positivos como negativos.



Cada una de las posiciones de impresión de la pantalla está constituida por 64 pixels.

sentencia **DRAW**, que permite dibujar líneas, y de la sentencia **CIRCLE** que dibuja circunferencias.
El conjunto de estas tres sentencias es suficientemente potente como para emprender la confección de diseños gráficos muy variados, sobre todo si pensamos en que puede hacerse uso de funciones ya conocidas como **INVERSE** y **OVER**, como complemento de las nuevas de manejo de la alta resolución.

La sentencia PLOT *redondea sus parámetros al entero más próximo, sumando 0.5, y calculando su parte entera.*

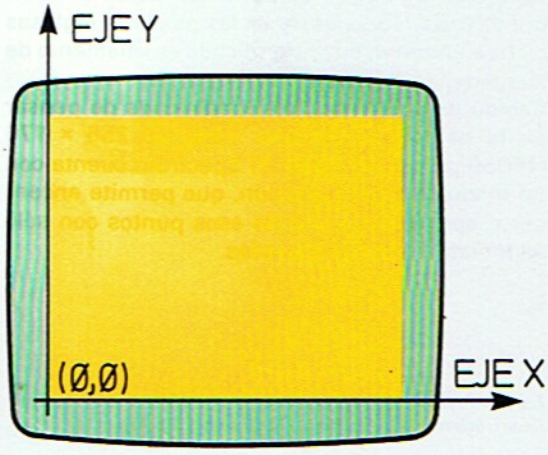


LA SENTENCIA PLOT

Por medio de **PLOT** puede encenderse un *pixel* cualquiera de la pantalla. El formato general de la sentencia es:

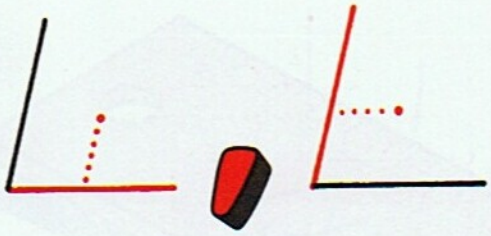
PLOT X,Y

El origen de coordenadas para la alta resolución se sitúa en el punto inferior izquierdo de la pantalla útil (descontadas las líneas del Sistema).

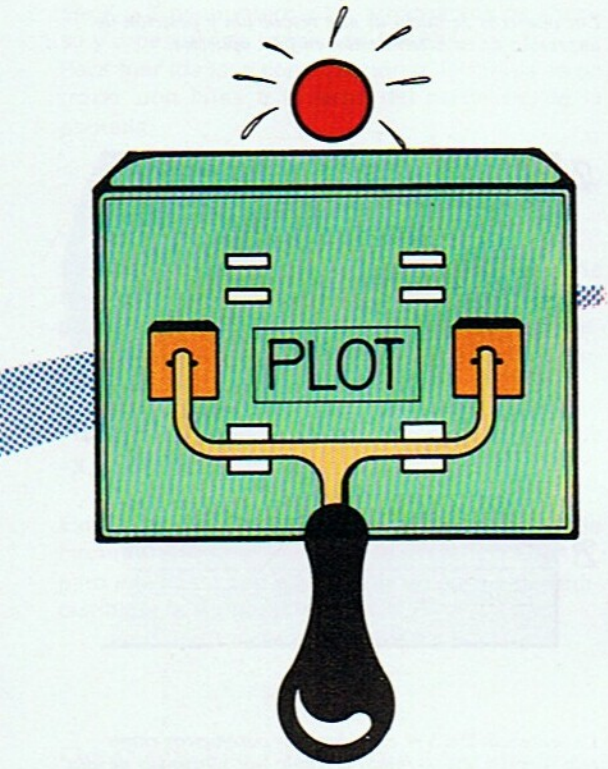


Donde **X** representa el desplazamiento en el eje horizontal, e **Y** en el vertical. Dado que el origen está situado en el extremo inferior izquierdo de la pantalla, es evidente que no pueden admitirse valores negativos. Asimismo, los máximos valores admisibles son 255 para la abscisa (EJE X) y 175 para la ordenada (EJE Y).
Uno de los cometidos más importantes de **PLOT** es la representación de funciones matemáticas en la pantalla de nuestro ordenador. El siguiente programa nos muestra la representación de las funciones **SIN** (seno) **COS** (coseno):

PLOT



El primer parámetro de PLOT *indica la coordenada X del punto a activar, y el segundo la coordenada Y.*



La sentencia PLOT *nos permite encender independientemente cualquier punto de la pantalla útil.*

```
  10 REM - GRAFICAS SENO Y COSENO
J. M. LOPEZ MARTINEZ
  20 FOR X=0 TO 255
  30 LET Y=88+88*SIN (X/128*PI)
  40 PRINT AT 21,0;"X=";X;" Y=";INT
(Y+.5);""
  50 GOSUB 100
  60 LET Y=88+88*COS (X/128*PI)
  70 GOSUB 100
  80 NEXT X
  90 STOP
 100 PLOT X,Y: RETURN
```

La sentencia **PLOT** admite, sin embargo, valores reales. En este caso se produce un redondeo automático al entero más próximo, labor que realiza el ordenador sumando .5 al valor y calculando su parte entera, de forma análoga a la descrita en la línea 40 del programa.

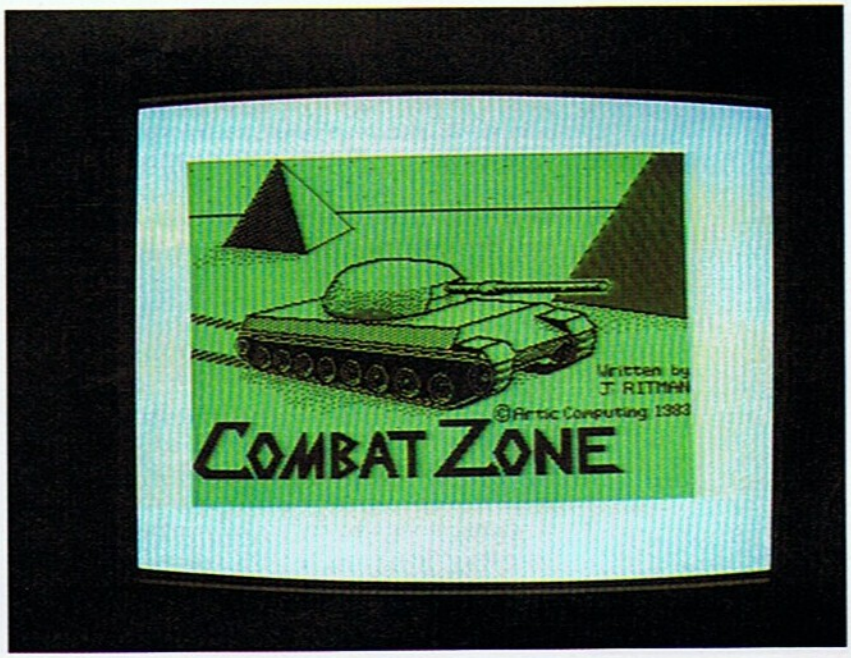
En cualquier caso, este hecho nos debe resultar familiar, puesto que se produce también cuando empleamos en la impresión convencional las funciones **TAB** y **AT** con números reales.

Basándonos en este hecho, podemos representar funciones en las cuales la diferencia entre los valores de la abscisa y la ordenada se hacen muy grandes, definiendo los incrementos del intervalo menores que la unidad, en nuestro ejemplo .5, para obtener una representación con una mayor densidad de puntos. A continuación veremos la representación de la función Y=X 2:

```
  10 REM - GRAFICA Y=X 2 - J.M. LOPEZ
MARTINEZ
  20 FOR X=-13 TO 13 STEP .5
  30 LET Y=X*X
  40 PLOT X+128,Y
  50 NEXT X
```

¡!

Cuando cada uno de los *pixels* de una pantalla puede ser activado independientemente, se dice que la pantalla es DIRECCIONABLE PUNTO A PUNTO. Por lo general, cuando la pantalla de un ordenador supera los 40.000 *pixels* direccionables independientemente se dice que posee ALTA RESOLUCION.



La sentencia **DRAW** *nos permite dibujar líneas en la pantalla.*

LA SENTENCIA DRAW

La sentencia **DRAW** permite dibujar líneas en la pantalla. Su formato general es: **DRAW X,Y**. Donde **X** especifica el valor de la abscisa e **Y** el de la ordenada. Sin embargo, al contrario de lo que sucede con **PLOT**, donde los argumentos son absolutos, **DRAW** tiene argumentos relativos, positivos o negativos, en base a la última posición accedida de la pantalla de alta resolución. Cuando se conecta el ordenador o se realiza un **CLS**, **CLEAR**, **RUN** o **NEW**; el puntero de impresión de la alta resolución se coloca al valor de (0,0), de forma similar a como lo hace el puntero de impresión convencional en (0,0).

Obviamente, la diferencia estriba en que el puntero de impresión de caracteres se sitúa en el extremo izquierdo de la línea superior de la pantalla, mientras que el de la alta resolución lo hace en el extremo izquierdo de la última línea directamente accesible (última de la fila 21).

A partir de esta situación inicial, y cada vez que se ejecuta una instrucción que contiene **PLOT**, **DRAW** o **CIRCLE**, el puntero se sitúa automáticamente en el último punto accedido, de forma similar a lo que sucede cuando imprimimos caracteres haciendo uso del punto y coma (;), aunque en este último caso, el posicionamiento se hace concretamente al siguiente carácter al impreso. En este estado de cosas, la sentencia **DRAW** traza una línea, de forma que el primero de sus extremos es el puntero actual de la alta resolución, y el segundo, uno situado tantas posiciones a la derecha o izquierda como indique el argumento **X** y tantas hacia arriba o abajo como indique el argumento **Y**.

Esto es posible porque los argumentos de la función **DRAW** pueden ser positivos o negativos. Cuando el argumento **X** es positivo, la abscisa del

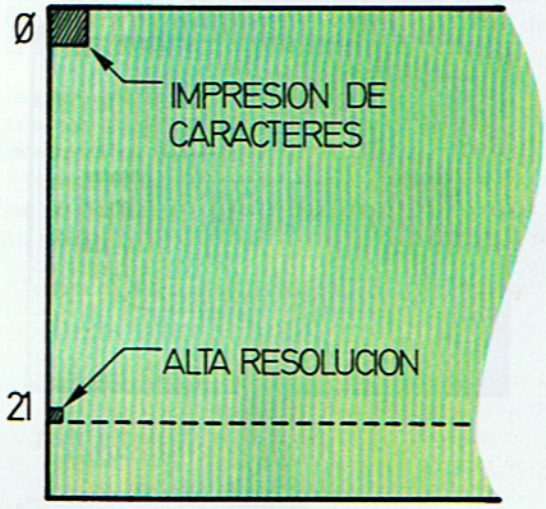
¡!

La sentencia **CIRCLE** nos permite el trazado de círculos en base a tres parámetros: coordenada central (eje X y eje Y) y radio.

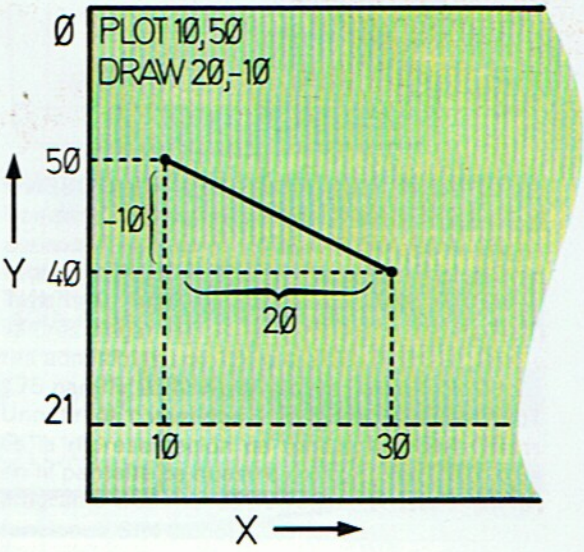
*

La función **POINT** tiene por parámetros la coordenada horizontal y vertical de un punto, y devuelve su estado (conectado=color de **INK** o desconectado=color de **PAPER**), siguiendo el convenio de 1 para conectado y 0 para desconectado.

Los punteros de inicio de alta resolución y posición de impresión de caracteres están en filas opuestas.



La sentencia **DRAW** *considera sus parámetros como relativos del último punto afectado por sentencias de alta resolución.*



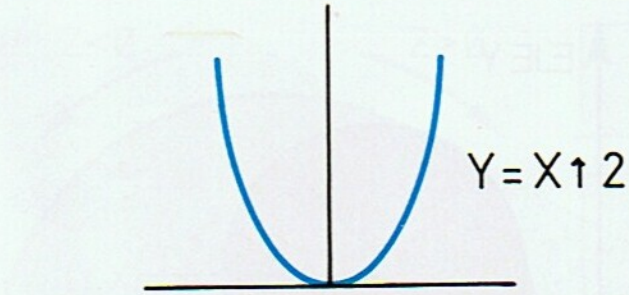
punto de destino se sitúa tantas posiciones a la derecha como indique el argumento, y si es negativo hacia la izquierda.
De forma similar, cuando el argumento **Y** es positivo la ordenada del punto de destino se sitúa tantas posiciones hacia arriba como indique el argumento, y si es negativo las mismas hacia abajo.
En definitiva, la sentencia **DRAW** traza la línea que une dos puntos. El primero de ellos el puntero de la alta resolución, y el segundo el situado en la posición suma algebraica (suma o diferencia según el signo) de los argumentos en el eje X e Y del puntero y los argumentos de abscisa y ordenada de la función **DRAW**.
Para fijar ideas, a continuación mostramos como trazar una línea a lo largo del perímetro de la pantalla:

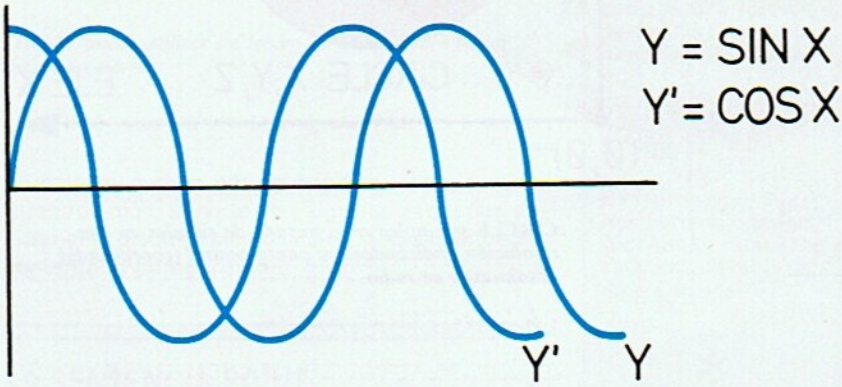
```
DRAW 255,0:DRAW 0,175:DRAW
-255,0:DRAW 0,-175
```

Basándonos en esta idea, podemos codificar una subrutina que permita dibujar rectángulos en la pantalla *(boxes)*, donde **A** y **B** serán las coordenadas de comienzo del dibujo y **X** e **Y** las longitudes de los lados del rectángulo:

```
PLOT A,B: DRAW X,0: DRAW 0,Y: DRAW
-X,0: DRAW 0,-Y: RETURN
```

Existe también la posibilidad de trazar arcos de circunferencia por medio de la sentencia **DRAW**, para ello basta con especificar un parámetro adicional de la forma: **DRAW X,Y,Z**.

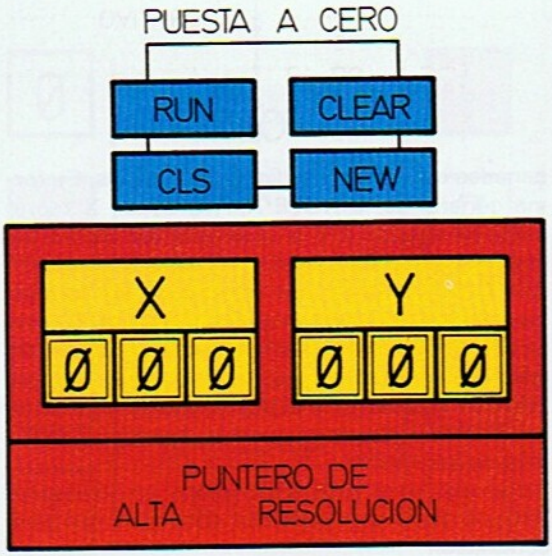


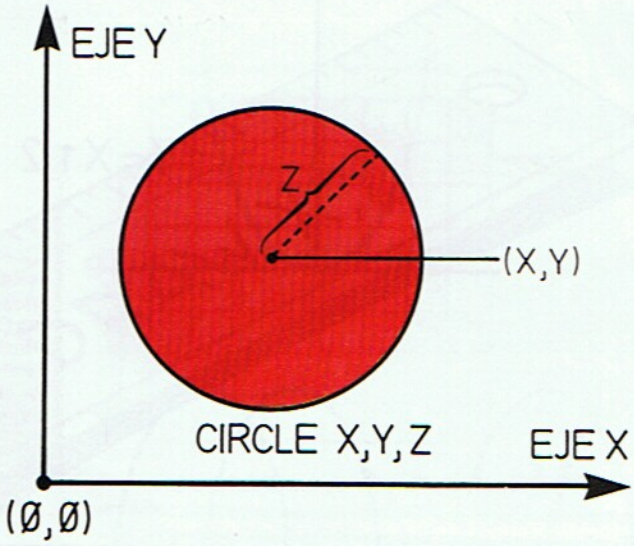


La sentencia **PLOT** *tiene una gran utilidad en la representación de funciones matemáticas.*

Donde **X** e **Y** son las coordenadas del punto de destino, y **Z** indica el número de radianes que debe girarse a medida que se avanza. Con un criterio similar al de la interpretación de **X** e **Y**, el parámetro **Z** implica un giro a la izquierda cuando es positivo y a la derecha en caso contrario.

CLS, CLEAR, RUN *y* **NEW** *ponen a cero el puntero de la alta resolución.*





CIRCLE *se emplea en el trazado de círculos en alta resolución, indicándose su punto central (coordenadas absolutas) y su radio.*

A continuación mostramos como trazar un cuarto de círculo a partir de la posición 128,88: **PLOT 128,88:DRAW 64,64,PI/2**.

Con todo lo dicho hasta ahora estamos en condiciones de trazar dibujos bastante complicados. ¡Vamos a ver como nos sale este!:

```
  10 REM - ANAGRAMA - J.M. LOPEZ MARTINE
Z
  20 INK 2
  30 PLOT 49,95: DRAW 30,0: DRAW 8,-8,-P
I/2: DRAW 0,-7: DRAW -8,-8,-PI/2: DRAW 8
,-8,-PI/2: DRAW -7,0: DRAW -8,7,PI/2: DR
AW -16,0: DRAW 0,-7: DRAW -7,0: DRAW 0,3
1
  40 PLOT 56,87: DRAW 19,0: DRAW 4,-4,-P
I/2: DRAW -3,-3,-PI/2: DRAW -20,0: DRAW
0,7
  50 PLOT 104,95: DRAW 7,0: DRAW 0,-15:
DRAW 8,-8,PI/2: DRAW 8,0: DRAW 8,8,PI/2:
 DRAW 0,15: DRAW 7,0: DRAW 0,-23: DRAW -
8,-8,-PI/2: DRAW -23,0: DRAW -7,8,-PI/2:
 DRAW 0,23
  60 PLOT 160,64: DRAW 0,23: DRAW 7,8,-P
I/2: DRAW 24,0: DRAW 8,-8,-PI/2: DRAW 0,
-23: DRAW -7,0: DRAW 0,15: DRAW -8,8,PI/
2: DRAW -8,0: DRAW -8,-8,PI/2: DRAW 0,-1
5: DRAW -7,0
```

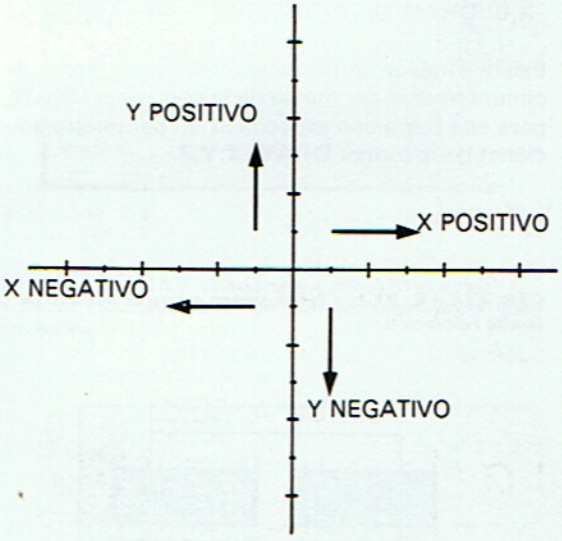
Los efectos conseguidos con **DRAW** pueden llegar a ser realmente curiosos, y si no lo creemos sólo tenemos que observar los siguientes ejemplos:

```
PLOT 90,80: DRAW 60,60,7689
PLOT 90,80: DRAW 60,60,8462
PLOT 90,80: DRAW 60,60,3572
PLOT 90,80: DRAW 60,60,1010
PLOT 90,80: DRAW 60,60,5387
PLOT 90,80: DRAW 60,60,3546
PLOT 90,80: DRAW 60,60,2547
PLOT 90,80: DRAW 60,60,4189
PLOT 90,80: DRAW 60,60,8888
```

LA SENTENCIA CIRCLE

La sentencia **CIRCLE** es la última de las encuadradas en el grupo de manejo de la alta resolución. Permite trazar círculos a partir de las coordenadas del centro y radio, especificados. Su forma general es: **CIRCLE X,Y,X,** donde **X** representa la abscisa e **Y** la ordenada del centro del círculo, siendo **Z** el radio.

La sentencia **DRAW** *admite tanto parámetros positivos como negativos.*



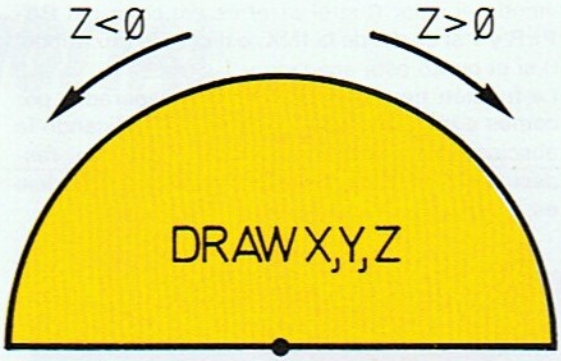
Con **CIRCLE**, como con el resto de las sentencias del grupo, puede emplearse el color. La única limitación reside en el hecho de que cuando un *pixel* se conecta en color, tanto de **INK** como de **PAPER**, además de a él mismo afecta al resto de los 64 *pixels* que componen su cuadrícula (posición de impresión).

En el siguiente programa vemos un «olímpico» ejemplo de lo que podemos conseguir con esta sentencia:

¡!

Las sentencias de alta resolución del tipo absoluto son **PLOT**, **CIRCLE**, y **POINT**, que por tanto, sólo admiten parámetros positivos.

```
  10 REM - AROS OLIMPICOS - J.M. LO
PEZ MARTINEZ
  20 FOR X=77 TO 177 STEP 50
  30 READ J: INK J
  40 FOR I=0 TO 2: CIRCLE X,98,30+I:
NEXT I
  50 NEXT X
  60 FOR X=102 TO 152 STEP 50
  70 READ J: INK J
  80 FOR I=0 TO 2: CIRCLE X,58,30+I:
NEXT I
  90 NEXT X
 100 DATA 1,0,2,6,4
```



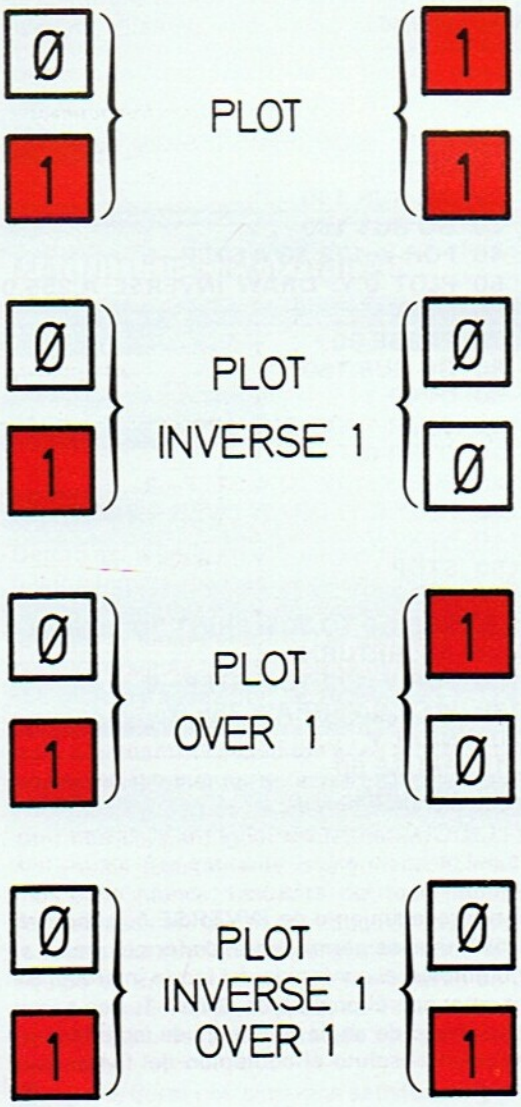
DRAW *puede utilizar un tercer parámetro para trazar arcos circulares.*

Ni más ni menos que la bandera olímpica ante nuestros ojos, aunque con algunas pequeñas deficiencias disculpables de color, debidas al hecho comentado anteriormente.

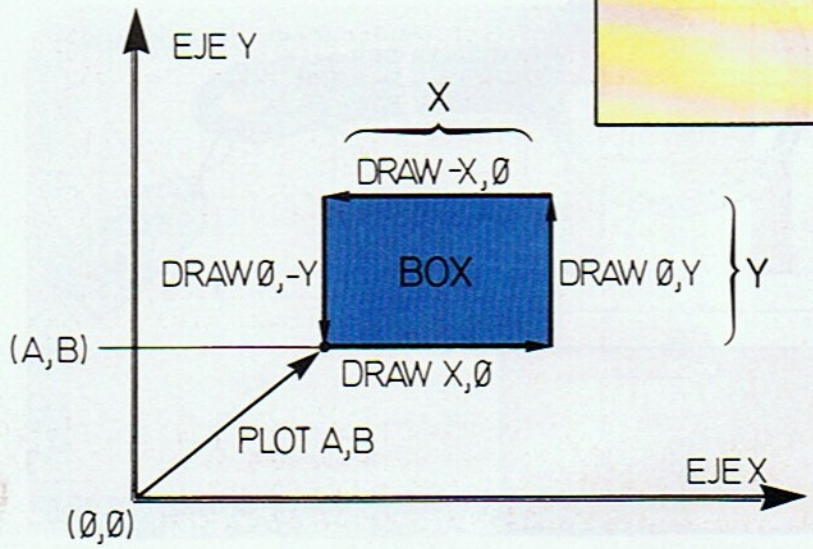
LA SENTENCIA POINT

Esta función suministra información sobre el color de un *pixel* determinado de la pantalla, devol-

En base a la sentencia **PLOT** *y utilizando los atributos* **INVERSE** *y* **OVER** *podemos manejar de todas las maneras posibles los puntos de la pantalla.*



Al rectángulo trazado en alta resolución en la pantalla se le denomina BOX.



¡!

El origen del puntero de alta resolución se encuentra en el *pixel* inferior izquierdo de la pantalla útil, perteneciente a la primera columna de la fila 23.

*

La sentencia **PLOT** nos permite controlar independientemente cada uno de los *pixels* de la pantalla.

*

La sentencia **DRAW** utilizada con sus dos primeros parámetros permite trazar líneas rectas; si se utiliza el tercer parámetro podemos obtener arcos circulares.

*

La sentencia **PLOT** se utiliza frecuentemente en la representación de funciones matemáticas.

¡!

Cualquier sentencia de alta resolución absoluta se puede convertir en relativa sumándole a las coordenadas X e Y, los **PEEK** de 23677 y 23678, respectivamente. De forma general, siendo X la coordenada horizontal e Y la vertical, un **PLOT** relativo podría ser **PLOT PEEK 23677+X, PEEK 23678+Y**; donde X e Y pueden adoptar valores positivos o negativos.

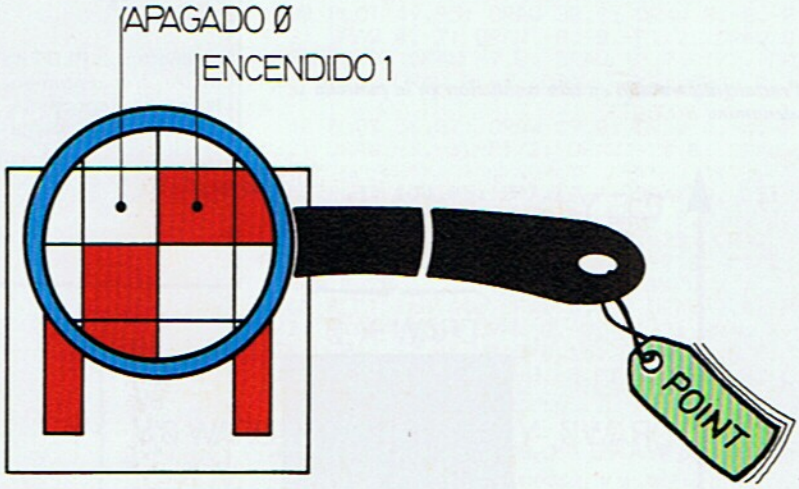
viendo el valor 0 si el *pixel* es del color del **PAPER** y 1 si es del de la **INK**; o dicho de otro modo: 0 si el punto está apagado y 1 si no.
La función tiene dos argumentos separados por comas e incluidos entre paréntesis, indicando la abscisa y ordenada del punto en concreto, respectivamente. El formato general de la sentencia es:

POINT(X,Y)

OTROS TRATAMIENTOS DE LOS *PIXELS*

Hasta ahora hemos hablado de como «encender» los puntos de la pantalla, sin comentar la forma de hacerlos desaparecer. Para ello existen diferentes posibilidades, dependiendo de lo que deseemos en cuestión.
PLOT INVERSE 1, se posiciona en el *pixel* indicado en el argumento y lo hace desaparecer, cambiando su color de **INK** por el del **PAPER**.
PLOT OVER 1, altera el estado del *pixel* a partir de su contenido actual, sea cual sea éste, es decir, si tenía el color de la **INK** pasa a tener el del **PAPER** y viceversa.
PLOT INVERSE 1 OVER 1, aparentemente no opera ningún cambio de estado en el *pixel*, aunque sí cambia el puntero de la alta resolución que quedará situado en las nuevas coordenadas.
La función desempeñada por **INVERSE** y **OVER**, para borrar lo dibujado por medio de las sentencias que conforman la alta resolución, hace que en muchas ocasiones sean sinónimos.
Sin embargo, sobre todo cuando se trata de dibujar sobre lo impreso, podemos ver más claramente la función que desempeña cada una de ellas. En el siguiente programa vemos un ejemplo claro, al borrar por dos procedimientos diferentes las líneas dibujadas sobre una pantalla impresa:

La función POINT *permite averiguar si un punto de la pantalla está encendido (color de* INK*) o apagado (color de* PAPER*).*



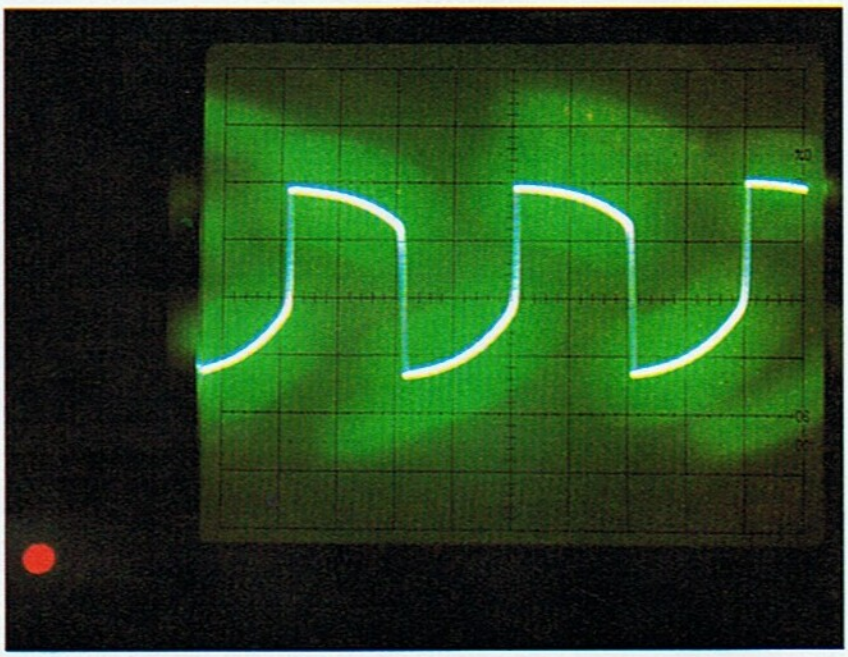
ATRIBUTO:
TINTA: ROJA
FONDO: BLANCO

```
 10 REM - SOBREIMPRESION - J.M. LOPEZ MARTINEZ
 20 GO SUB 160
 30 GO SUB 180
 40 FOR Y=172 TO 4 STEP—8
 50 PLOT 0,Y: DRAW INVERSE A;255,0
 60 NEXT Y
 70 PAUSE 50
 80 GO SUB 160
 90 OVER 1
100 OVER 1: GO SUB 180
110 OVER 0
120 FOR Y=172 TO 4 STEP—8
130 PLOT 0,Y: DRAW OVER 1;255,0
140 NEXT Y
150 STOP
160 CLS
170 FOR I=0 TO 703: PRINT "O";: NEXT I: PAUSE 50: RETURN
180 FOR Y=172 TO 4 STEP—8
190 PLOT 0,Y: DRAW 255,0
200 NEXT Y
210 PAUSE 50
220 RETURN
```

Por el procedimiento de **INVERSE 1**, el borrado de las líneas es permanente, de forma que se altera también el contenido del fondo impreso. Sin embargo, con el empleo de **OVER 1**, nos permitimos el lujo de anular el dibujo de las líneas sin alterar en absoluto el contenido del fondo de la pantalla.

GRABACION DIGITAL

HEMOS comentado ampliamente en las páginas anteriores, las condiciones que deben reunir tanto las grabadoras como las cintas magnéticas empleadas en los procesos de transferencia de datos entre Spectrum y casete.
Analizaremos ahora la forma de comunicación utilizada por nuestro micro, para enviar y recibir información a través del casete, así como todos los circuitos internos que intervienen para llevar a buen término el proceso.
A estas alturas suponemos que estará suficientemente claro que el vocabulario manejado por nuestro Spectrum no va más allá de distintas combinaciones de unos y ceros (bits), agrupados de ocho en ocho (bytes). Estos últimos, combinados en cierto orden, conforman frases más o menos largas, las cuales denominamos programas.

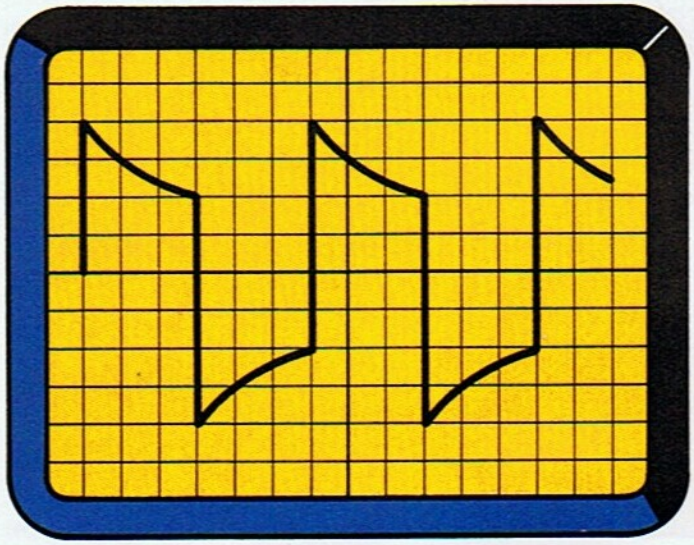


Las tensiones correspondientes al uno y cero lógicos deben estar lo suficientemente separadas para evitar errores en la interpretación.

MAGNITUDES ANALOGICAS

Dentro del mundo en el cual nos desenvolvemos, habitualmente realizamos ciertas medidas de algunas magnitudes de uso cotidiano, como la velocidad, la temperatura o, por ejemplo, la altura de las personas. Es fácil escuchar frases como: «El calor es sofocante. Al menos hay cuarenta grados a la sombra».
Pero efectuar un cálculo tan poco técnico supondrá, inevitablemente, un error bastante elevado. Podemos ayudarnos de un pequeño termómetro barométrico, pero igualmente, determinar si son 40 grados exactamente o algunas decimas de más o de menos, resultará del todo imposible. Aconsejados por algún maniático de la precisión, tal vez podamos tener acceso a un modernísimo equipo de los empleados en los centros de predicción meteorológica. Estos aparatos proporcionan medidas de la temperatura con varias cifras decimales.
Entonces, quizá nos sintamos satisfechos por co-

Aspecto de la señal tomada de EAR *en la salida de casete.*



¡!

Magnitudes analógicas son aquellas que crecen y decrecen de manera constante.

*

Magnitudes digitales son aquellas que crecen y decrecen por valores constantes y de forma discontínua.

*

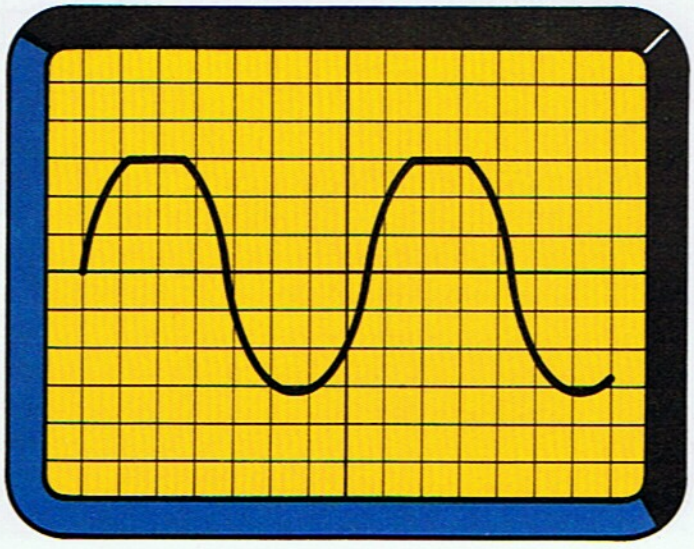
Los unos y los ceros deben ser claramente distinguidos en todos los procesos por el ordenador; de ahí que se empleeen magnitudes digitales, claramente diferenciadas en los dos valores.

A estas alturas suponemos que estará suficientemente claro, que el vocabulario manejado por nuestro Spectrum no va más allá de distintas combinaciones de unos y ceros (bits).

nocer la temperatura exacta reinante en ese momento. La primera consecuencia que debemos extraer, es que entre dos temperaturas se pueden dar infinitos estados diferentes. Es decir, uno de estos aparatos de precisión habrá detectado como al bajar y subir la temperatura lo hace de forma continua, atravesando infinitas temperaturas intermedias.

Y a esto es a lo que se denomina magnitud analógica, es decir, aquella que crece y decrece de forma continuada. ¿Pero qué demonios nos importa a nosotros si eran 39 ó 40 ó 40,745 grados?

Aspecto de la señal de entrada a la U.L.A. (pin 28) y salida hacia el casete tras el interface.



MAGNITUDES DIGITALES

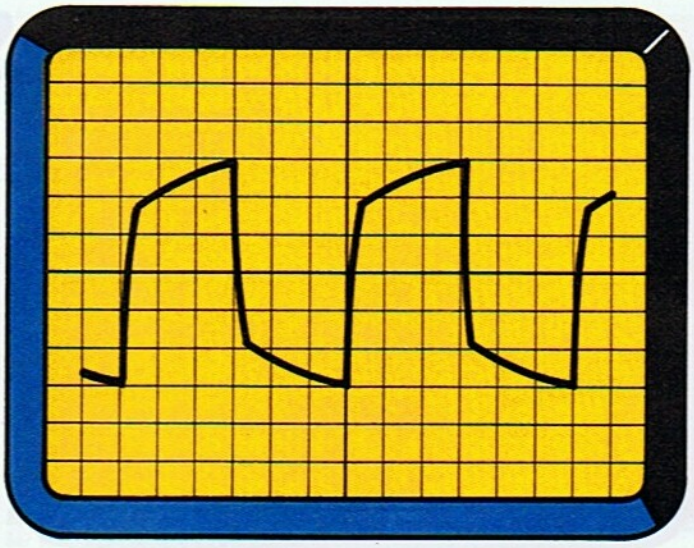
Puede que en temas como la temperatura no nos importe nada la precisión, y mucho menos lle-

gando al extremo de los decimales; sin embargo, para su correcto funcionamiento, el Spectrum no se puede permitir el lujo de la inexactitud en los procesos.

El funcionamiento de nuestro Spectrum está basado en la serie de impulsos eléctricos que circulan a través de sus circuitos. Como entre los cables y componentes electrónicos no pueden viajar ni palabras, ni marcianitos, sino sólo señales eléctricas, éstas deben de ser las menos posibles, con el objeto de evitar una posible confusión.

Esta es la razón por la cual las palabras de su vocabulario están siempre formadas por unos y ceros: es más fácil detectar dos tensiones que 256 diferentes, y para él debe quedar claro en todo momento cuál corresponde a un uno y cual a un cero. Además protestará enérgicamente, o lo que es lo mismo, no se enterará de nada, si intenta evaluar las tensiones que circulan entre sus circuitos de la misma manera que lo hacíamos con las magnitudes analógicas; es decir, creciendo y decreciendo de manera constante, pasando por infinitos estados intermedios.

Por ello, las tensiones correspondientes al uno y cero lógicos deben estar lo suficientemente separadas para evitar errores en la interpretación; por tanto, deben crecer o decrecer por impulsos de valor constante y forma discontinua, sin infinitos estados intermedios. Son lo que se denominan magnitudes o señales digitales: entre dos estados distintos, se produce siempre un salto fijo y determinado.



Cero lógico obtenido en el pin 28 de la U.L.A.

El tiempo es una magnitud continua (analógica); a pesar de ello, su medida puede ser tomada con aparatos analógicos o digitales.

MEDIDA ANALOGICA

MEDIDA DIGITAL

Lo mismo debe ocurrir con los sonidos que el Spectrum envía hacia el casete o recibe de éste. El proceso es siempre el mismo. Durante la grabación, las señales eléctricas deben convertirse en impulsos sonoros, y al contrario mientras dure la carga de un programa almacenado en cinta. Concretando, la frecuencia de las señales sonoras, o lo que es lo mismo, el tiempo que están activas, debe ser diferente para el cero y el uno lógico.

Como es habitual, estas frecuencias deben ser específicas y estar lo suficientemente separadas, de manera que no pueda caber el error de interpretación. La U.L.A. es la encargada de gestionar todo el proceso

TRANSMISION AL CASETE

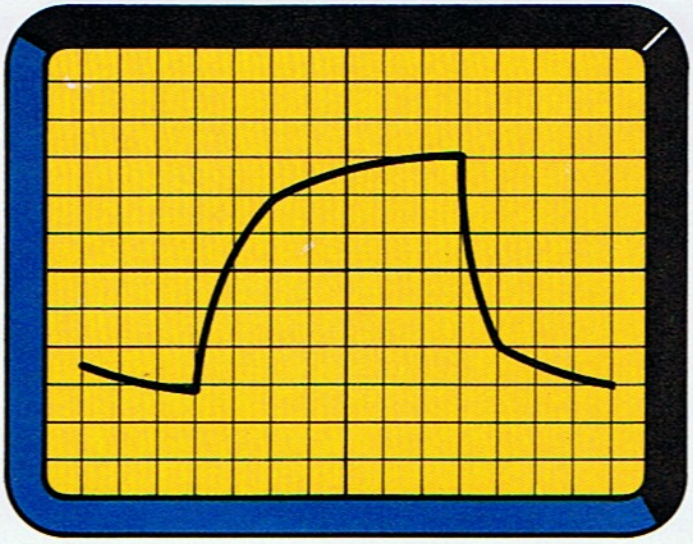
El proceso de transmisión de datos entre Spectrum y casete, se gestiona satisfactoriamente mediante una parte del *software* contenido en la R.O.M., denominado genéricamente *Cassette Handling Routines* (Rutinas de manejo del casete), que están comprendidas entre las direcciones

¡!

Esta misma distinción clara debe cumplirse entre los sonidos que el Spectrum recibe y emite desde y hacia el casete.

*

El proceso de transmisión de datos entre el Spectrum y el casete, se controla mediante una zona del *firmware* denominada *Cassette Handling Routines* (Rutinas de Manejo del Casete).

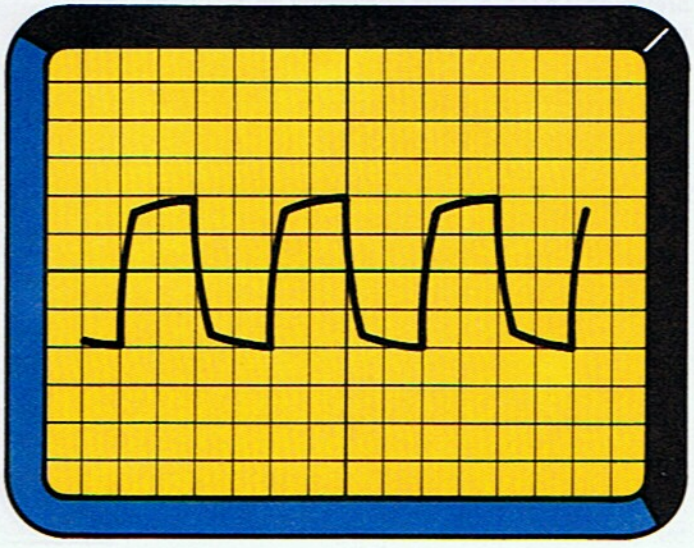


Uno lógico a la salida del pin *28.*

decimales 1218 y 2547. Este se encarga de descomponer cada byte en sus ocho bits respectivos, para ser enviados uno tras otro hacia la grabadora.

A través del *pin* (patilla) 28 de la U.L.A. *(Uncommited Logic Array)* salen en serie, a una velocidad media de 1500 baudios (bits/seg). Para evitar posibles confusiones, la frecuencia de los unos y los ceros es distinta. Como podemos observar en la fotografía tomada de la pantalla de un osciloscopio, la frecuencia del cero, del orden de los 2000 Hertzios, es doble de la del uno, aproximadamente 1000 Hertzios.

*Aspecto de la onda producida por la cabecera de transmisión (*pin *28).*



El tono guía *(leader)* que emite el Spectrum previamente al envío de los datos, tiene por objeto elevar al nivel correcto el control del volumen de grabación en los casetes que lo tienen automático, con el fin de evitar saturaciones en la cinta por ser el volumen demasiado alto, o impedir que las grabaciones se realicen a niveles sonoros bajos que imposibiliten su reconstrucción.

Sobre la información contenida en la cabecera, tras el tono guía, nos remitimos al comentario que se hace sobre el tema en el programa INDEX. El *leader* no puede ser nunca confundido con un bloque de información, dado que su frecuencia es menor que la de uno, y por tanto que la del cero.

En las fotografías, observamos el aspecto de las señales digitales que la U.L.A. envía hacia el casete. Estas tienen forma de pulso casi cuadrado, pero el casete no es capaz de manejar señales digitales, debido a lo cual es necesario disponer de un interface entre ambos, que sea capaz de mediar en el proceso.

Otra característica a tener en cuenta es la ausencia de tiempos muertos entre la señal que compone un impulso y el siguiente. Gracias a ello, se evita la posibilidad de interferencia por ruido eléctrico, asociada a todo proceso en el cual se utilice la cinta magnética, y la inevitable profusión de errores en la interpretación, que por esta causa se podrían provocar.

Los resultados obtenidos usando el osciloscopio, muestran que el interface amplifica ligeramente la señal procedente de la U.L.A., a la vez que adapta impedancias (resistencia) de entrada/salida entre Spectrum y casete.

A través del *pin* 28, sale también la señal que activa el pequeño minialtavoz *(buzzer)* cuando ejecutamos, por ejemplo, una orden **BEEP**. Durante las operaciones en las cuales interviene el buzzer, el nivel de tensión de la patilla 28 está sobreelevado con el fin de que este suene, no haciéndolo mientras ejecutamos una orden **SAVE**.

Durante el proceso de carga de un programa, a través del interface llega a la U.L.A. la señal proviniente del casete. Esta va interpretando las distintas frecuencias transformándolas en unos y ceros lógicos. Hemos comprobado conforme se aumenta el volumen de la señal, que éste alcanza para un casete de calidad normal el punto óptimo en los tres cuartos, aproximadamente, del volumen total.

Al accionar sobre el control de tono, se produce una atenuación de la onda trasmitida, si éste no ha sido colocado al máximo de agudos. Igualmente, al manejar casetes estéreo, el menor desajuste en las cabezas provoca distorsiones en las señales transmitidas.

MOTOS DE LUZ

SEGURAMENTE, muchos de nosotros hemos oído hablar, o hemos llegado a ver la película TRON; un homenaje del Séptimo Arte al mundo de la informática.

El protagonista de su argumento es un competente programador: Flynn, que a causa de las artimañas del malvado CCP (Control Central de Programas), se materializa en el interior del ordenador de una gran empresa de *software* para videojuegos, intentando descubrir al «pirata» que le espía sus creaciones.

Dentro de la enorme máquina, convertido en una instrucción, va en la búsqueda de la instrucción TRON, que envió poco antes de ser desintegrado por el CCP, para realizar una investigación. En su recorrido por los buses, memorias, *ports*, etc... Flynn se ve amenazado por sus propias creaciones, debiendo competir por su existencia en el duelo de los aros de energía, o en la mortal carrera de las motos de luz.

MOTOS DE LUZ

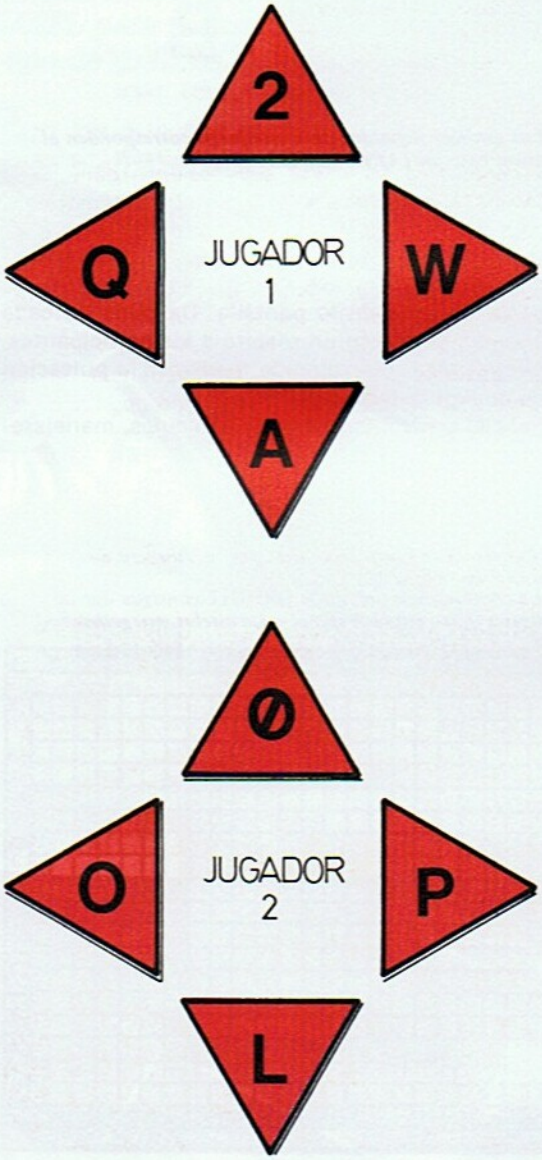
Nuestro programa recrea en la pantalla del Spectrum el duelo que Flynn y el CCP protagonizan en las fantásticas motos de luz. El tablero de juego es una malla que ocupa la práctica totalidad de la pantalla, rodeada de unos muros de energía, con los cuales el choque es mortal.

El juego necesita, evidentemente, el concurso de dos participantes, los cuales guiarán sus motos de luz intentando que el contrincante se estrelle contra algún muro de energía.

Estos especiales vehículos van dejando a su paso un sólido rastro energético, de forma que debemos intentar cercar a nuestro oponente, para conseguir que se choque, ya sea contra nuestro haz de energía, contra los muros, o contra su propia estela energética; naturalmente, todo ello antes de que nosotros cometamos alguno de los mencionados errores.

El vencedor del juego es aquel que consigue llegar a diez puntos antes que el contrario, teniendo en cuenta que cada choque proporciona un punto al superviviente. La posición de inicio de

En el gráfico se detallan las teclas para el movimiento de los vehículos de cada uno de los jugadores.



Los caracteres que aparecen en el listado con un subrayado simple, deben ser introducidos como los caracteres gráficos de las teclas correspondientes.

*

Los caracteres que figuran doblemente subrayados corresponden a los gráficos cambiados, es decir, los que se obtienen pulsando simultáneamente **CAPS SHIFT** y la tecla afectada, en el modo **GRAPHICS**.

*

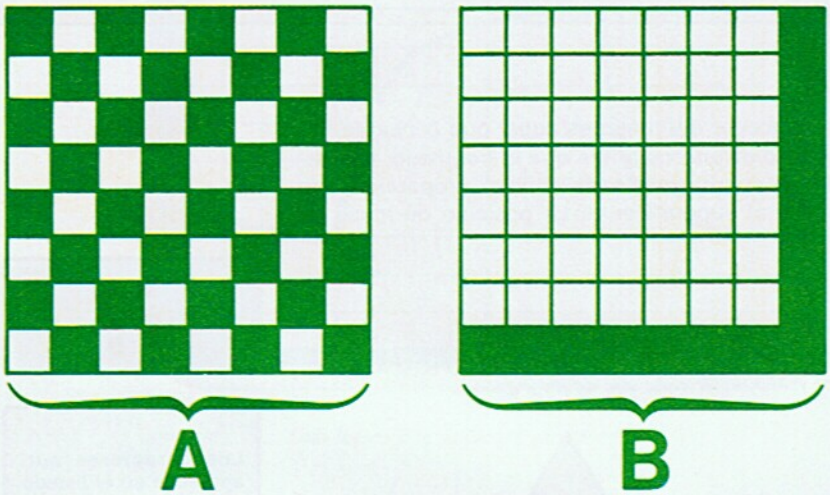
Para la toma de datos se ha recurrido a la lectura directa del teclado mediante la función **IN**.

*

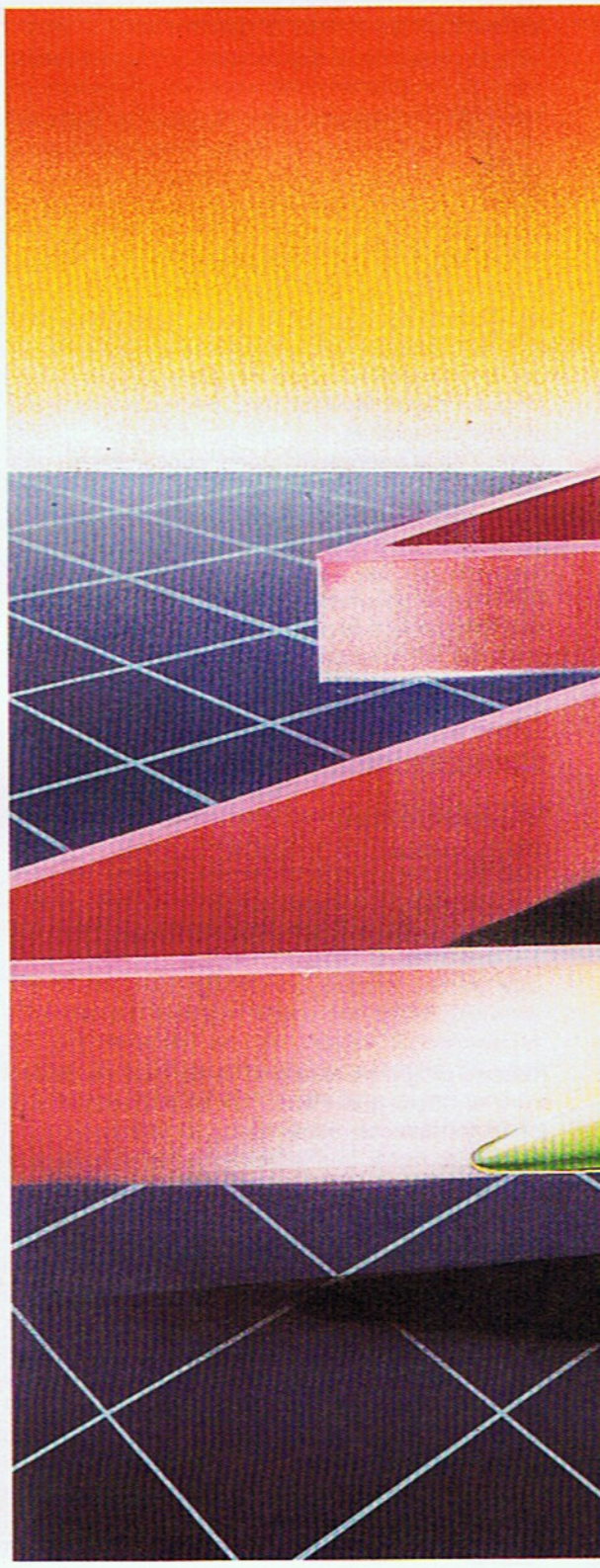
Para la grabación del programa procederemos del modo habitual, mediante el comando **SAVE "MOTOS LUZ"**, o bien, con la opción de autoejecución **SAVE "MOTOS LUZ" LINE 10**.

cada juego es siempre la misma: el jugador uno, que manejará la parte izquierda del teclado, aparece en la fila diez y columna cuatro de la pantalla, y su oponente, en la misma fila, pero en la parte derecha de la pantalla. Después de cada choque, se ofrece un respiro a los participantes, el cual será interrumpido mediante la pulsación de cualquier tecla.

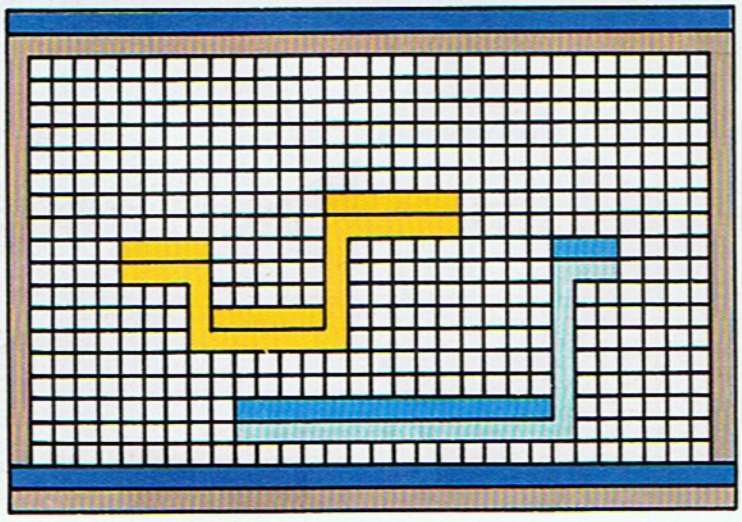
Para el control de nuestros vehículos, manejaremos cuatro teclas: el jugador uno, utilizará las teclas 2, Q, W y A, para arriba, izquierda, derecha y abajo, respectivamente; su oponente, controlará las teclas 0, O, P y L, con los mismos significados respectivos. A este respecto, las motos co-



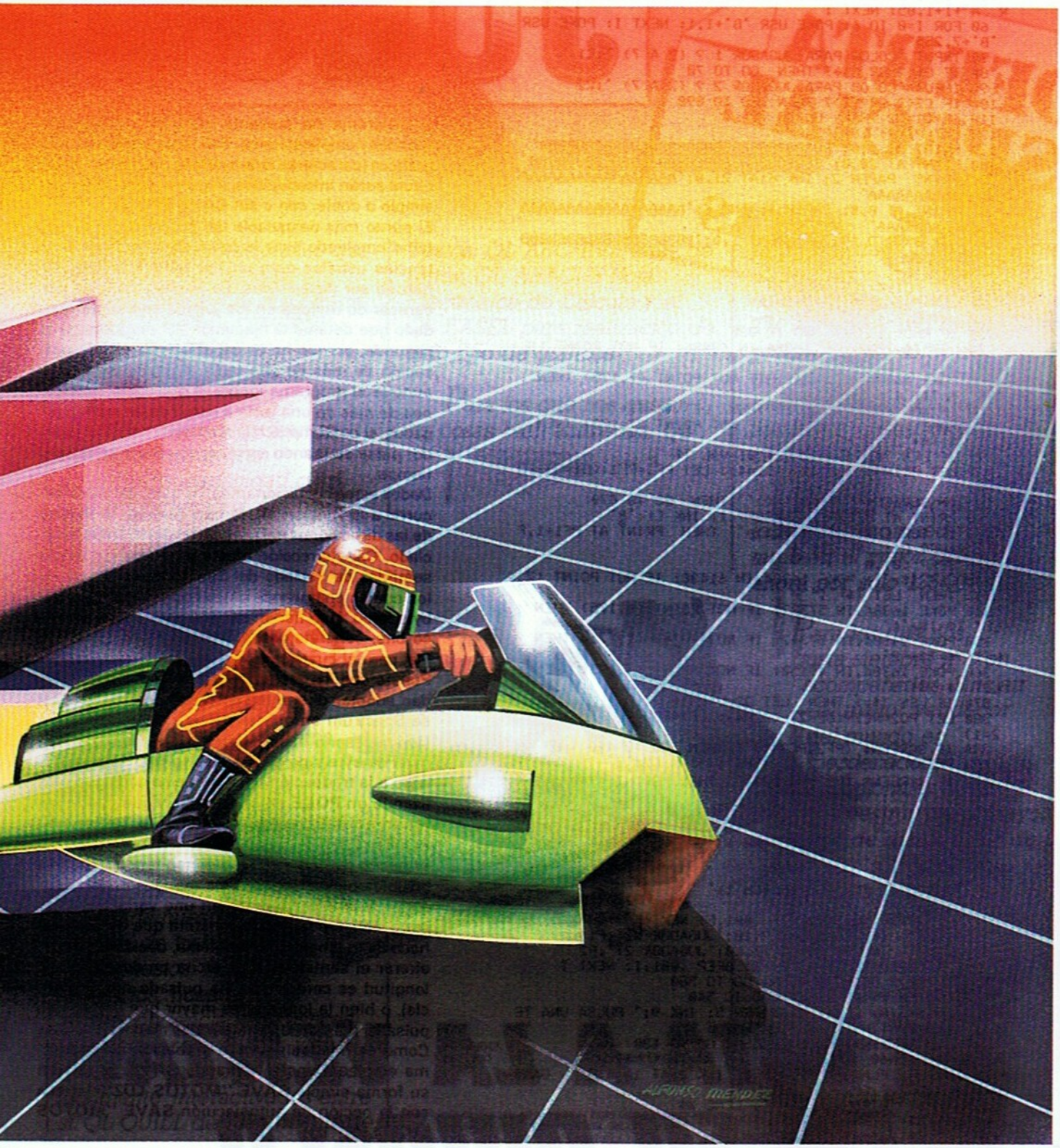
Los gráficos definidos para este juego corresponden al muro exterior (A) y al mallado (B).



La combinación del modo BRIGHT *consigue dar un ligero efecto tridimensional a las estelas energéticas.*



mienzan siempre con un mismo sentido de desplazamiento, que no alteran hasta que se pulsa una y sólo una de las teclas a tal fin.
Antes de cada partida podremos elegir los colores de nuestras motos de luz, comprendidos entre el dos (rojo) y el siete (blanco), dado que el cero y el uno (negro y azul) serían difícilmente distinguibles sobre el tablero, y siempre y cuando cada jugador escojamos un color diferente, para evitar confusiones.



```
 10 REM ************************
 20 REM * J.M. MAYORAL SERRANO *
 30 REM ************************
 40 PAPER 1: BORDER 1: INK 7: CLS : POKE 23609,30: P
OKE 23562,1
 50 FOR I=0 TO 7 STEP 2: POKE USR "A"+I,170: POKE US
R "A"+I+1,85: NEXT I
 60 FOR I=0 TO 6: POKE USR "B"+I,1: NEXT I: POKE USR
 "B"+7,255
 70 INPUT "COLOR PARA JUGADOR 1 ? (2 A 7) ";C1
 80 IF C1<2 OR C1>7 THEN  GO TO 70
 90 INPUT "COLOR PARA JUGADOR 2 ? (2 A 7) ";C2
100 IF C2<2 OR C2>7 THEN  GO TO 090
110 IF C1=C2 THEN  GO TO 70
120 LET P1=0: LET P2=0
130 PRINT AT 1,0;"88888888888888888888888888888888"
140 PRINT AT 20,0;"88888888888888888888888888888888"
150 PRINT  PAPER 2; INK 6;AT 21,0;"AAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAA"
160 PRINT AT 0,0; PAPER 2; INK 6;"AAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAA"
170 FOR I=2 TO 19: PRINT AT I,0;"8888888888888888888
888888888888"
180 NEXT I
190 LET PC1=4: LET PF1=10
200 LET PC2=27: LET PF2=10
210 LET D1=2: LET D2=4
220 IF RND<.5 THEN  GO TO 330
230 LET T$="": POKE 16384,IN 63486: IF NOT POINT (6,
175) THEN  LET T$="1"
240 POKE 16384,IN 64510: IF NOT POINT (7,175) THEN
LET T$=T$+"4"
250 IF NOT POINT (6,175) THEN  LET T$=T$+"2"
260 POKE 16384,IN 65022: IF NOT POINT (7,175) THEN
LET T$=T$+"3"
270 IF LEN T$=1 THEN  LET D1=VAL T$
280 LET PC1=PC1+(D1=2)-(D1=4): LET PF1=PF1+(D1=3)-(D
1=1)
290 IF SCREEN$ (PF1,PC1)<>"" THEN  GO TO 470
300 PRINT AT PF1,PC1; BRIGHT 1; INK C1;"8"
310 IF SCREEN$ (PF1+1,PC1)="" THEN  PRINT AT PF1+1,P
C1; PAPER C1; INK C1;"A"
320 BEEP .01,0
330 LET T$="": POKE 16384,IN 61438: IF NOT POINT (7,
175) THEN  LET T$="1"
340 POKE 16384,IN 57342: IF NOT POINT (6,175) THEN
LET T$=T$+"4"
350 POKE 16384,IN 57342: IF NOT POINT (7,175) THEN
LET T$=T$+"2"
360 POKE 16384,IN 49150: IF NOT POINT (6,175) THEN
LET T$=T$+"3"
370 IF LEN T$=1 THEN  LET D2=VAL T$
380 LET PC2=PC2+(D2=2)-(D2=4): LET PF2=PF2+(D2=3)-(D
2=1)
390 IF SCREEN$ (PF2,PC2)<>"" THEN  GO TO 440
400 PRINT AT PF2,PC2; BRIGHT 1; INK C2;"8"
410 IF SCREEN$ (PF2+1,PC2)="" THEN  PRINT AT PF2+1,P
C2; PAPER C2; INK C2;"A"
420 BEEP .01,10
430 GO TO 230
440 PRINT AT PF2,PC2; FLASH 1;" "
450 LET P1=P1+1
460 GO TO 490
470 PRINT AT PF1,PC1; FLASH 1;" "
480 LET P2=P2+1
490 FOR F=30 TO 60: BEEP .001,F: NEXT F
500 PRINT  INVERSE 1;AT 9,10;"JUGADOR 1:";P1
510 PRINT  INVERSE 1;AT 11,10;"JUGADOR 2:";P2
520 FOR I=60 TO 40 STEP -1: BEEP .001,I: NEXT I
530 IF P1>9 OR P2>9 THEN  GO TO 580
540 IF INKEY$<>"" THEN  GO TO 540
550 PRINT )0; FLASH 1; PAPER 5; INK 9;" PULSA UNA TE
CLA PARA CONTINUAR "
560 IF INKEY$<>"" THEN  CLS : GO TO 130
570 GO TO 560
580 PRINT  FLASH 1; PAPER 6; INK 2;AT 13,4;"    GANA
EL JUGADOR ";(P2=10)+1;"  "
590 INPUT "OTRO JUEGO ? "; LINE X$
600 IF X$="" THEN  GO TO 590
610 IF X$(1)<>"N" THEN  RUN
```

EL PROGRAMA

El programa no presenta para su introducción grandes complicaciones. Tan solo aparecen unos gráficos (caracteres subrayados), que como es habitual serán introducidos, según el subrayado sea simple o doble, con o sin **CAPS SHIFT**.

El punto más destacable del programa es el sistema empleado para la toma de datos. Las sentencias usuales para tal fin: **INPUT** e **INKEY$** han de ser desechadas. La primera de ellas por carecer de utilidad en los juegos de tiempo real, dado que detiene la ejecución del programa; y la segunda, porque no es capaz de registrar la pulsación de más de una tecla al mismo tiempo, y aunque el programa ignore la pulsación simultánea de más de una tecla a cargo de un mismo jugador, si contempla, lógicamente, que cada jugador pulse su mando correspondiente a un mismo tiempo.

Dados estos condicionantes, ha sido necesario recurrir a la función **IN**, la cual permite la lectura de las direcciones del *port* correspondientes al teclado. Cada dirección afecta a una de las veinte semifilas del teclado del Spectrum, cada una de los cuales comprende cinco teclas; así pues, del byte leído, sólo tienen sentido para nosotros cinco bits, concretamente los menos significativos (del cero al cuatro).

Puesto que los restantes bits del byte leído pueden adoptar diferentes valores, según la versión de Spectrum en que se ejecute el programa, ha sido necesario hacer un análisis del byte no a su nivel de byte sino de bit, para lo cual se ha recurrido a la traducción directa que se consigue realizando un **POKE** en la pantalla, y que puede ser fácilmente investigada mediante la función **POINT**.

El hecho de que se vayan consultando todas las posibles pulsaciones de los mandos de cada jugador, y almacenándolos en una variable de cadena (**T$**), se debe a la consulta que después se hace de la longitud de la misma, decidiéndose no alterar el sentido de la marcha en dos casos: la longitud es cero (no se ha pulsado ninguna tecla), o bien la longitud es mayor que uno (se ha pulsado más de un mando del mismo jugador).

Como es habitual, para la grabación del programa emplearemos el comando **SAVE**, ya sea en su forma simple **SAVE "MOTOS LUZ"**, ya sea con la opción de autoejecución **SAVE "MOTOS LUZ" LINE 10.**